Blind Packet Forwarding

A Clean-Slate Security Approach for Future Networks

D I S S E R T A T I O N

to obtain the academic grade
doctor rerum naturalium
(dr. rer. nat.)
in Computer Science

Submitted to the
Faculty of Economics
Institute for Computer Science and Business Information Systems
University of Duisburg-Essen

by
Irfan Simsek
born on 04.08.1983 in Kayseri, Turkey

President of the University of Duisburg-Essen:
Prof. Dr. Ulrich Radtke

Dean of the Faculty of Economics:
Prof. Dr.-Ing. Erwin P. Rathgeb

Reviewers:

1. Prof. Dr.-Ing. Erwin P. Rathgeb
2. Prof. Dr. Paul Müller

Submitted on:	August 11, 2016
Date of Disputation:	October 26, 2016

Selbständigkeitserklärung

Hiermit erkläre ich, die vorliegende Arbeit selbständig ohne fremde Hilfe verfaßt und nur die angegebene Literatur und Hilfsmittel verwendet zu haben.
Weiter erkläre ich, dass diesem Promotionsverfahren keine Promotionsversuche in diesem Fach oder in einem anderen Fach vorausgegangen sind und dass die eingereichte Arbeit oder wesentliche Teile derselben in keinem anderen Verfahren zur Erlangung eines akademischen Grades vorgelegt worden sind.

Irfan Simsek
August 11, 2016

Abstract

Meanwhile, there exist a wealth of approaches for a Future Network Architecture (FNA). Although these approaches differ in their orientation, they all suggest that a network should be service-oriented and flexibly orchestrated from atomic smart in-network services. In order to utilise the complete functionality of the orchestrated network, the in-network services require access to various control data that is exchanged in different ways. Hence, the communication endpoints have to expose more and more information about themselves. However, the in-network services as well as third parties are able to sniff information while it is transferred in cleartext. Beside these considerations, end-to-end encryption is the de facto method applied to provide information confidentiality for two communicating endpoints. But if the communicating endpoints perform end-to-end encryption, in-network services cannot accomplish their tasks anymore, since they cannot access the encrypted control data. Thus, it becomes impossible to fully utilise the benefits of FNA approaches.

These issues indicate that it is only possible to realise one of the two goals – information confidentiality and smart in-network services – at once. But we demonstrate the feasibility to simultaneously establish smart in-network services and to provide information confidentiality by redesigning the packet forwarding service to make it operate blindly, which we call Blind Packet Forwarding (BPF). We choose this in-network service as an example because packet forwarding is one of the basic services required for most network architectures. Moreover, packet addresses act as the basis for operations performed by further in-network services. Furthermore, it was not possible so far to transfer packet addresses in end-to-end encrypted form. BPF provides confidentiality for packet addresses during transmission as well as during processing by network nodes.

Keywords:
Blind Packet Forwarding, Network Address Confidentiality, Future Network Architecture, Public key Encryption with Keyword Search, Locator/Identifier Split, OpenFlow

Acknowledgements

First and foremost, I would like to express my sincere gratitude to my primary advisor Prof. Dr.-Ing Erwin P. Rathgeb for the continuous support of my PhD study and related research in the Computer Networking Technology Group at the University of Duisburg-Essen. I cannot thank him enough for his patience, motivation, and invaluable comments. I would also like to thank my secondary advisor Prof. Dr. Paul Müller for his thesis review.

I will always be grateful to my former colleagues Prof. Dr. Martin Becke and Hakim Adhari for their team spirit, help, and willingness for discussion. Moreover, I would like to express my special thanks to all members of our group, in particular Julius Flohr and Nihad Cosic, for the continuous support and pleasant working atmosphere.

Furthermore, I would like to thank my parents Güler and Gazi Simsek as well as my brother Emre Simsek, for all their love and encouragement. Last but not the least, I will be forever grateful to my wife Sema Simsek for her continuous support, patience, and love.

Contents

Glossary

AES	Advanced Encryption Standard
BGP	Border Gateway Protocol
BNS	Blind Network Stack
BPF	Blind Packet Forwarding
BPF – GLI	Blind Packet Forwarding in Global Locator, Local Locator, and Identifier Split
BPF – HAIR	Blind Packet Forwarding in Hierarchical Architecture for Internet Routing
BPF – HiLLIS	Blind Packet Forwarding in Hierarchical Level-based Locator/Identifier Split
CA	Certification Authority
CMS	Core Mapping Service
DHCP	Dynamic Host Configuration Protocol
DNS	Domain Name System
DPI	Deep Packet Inspection
ESP	IP Encapsulating Security Payload
FNA	Future Network Architecture
GLI – Split	Global Locator, Local Locator, and Identifier Split
GMS	Global Mapping System
HAIR	Hierarchical Architecture for Internet Routing
HTTP	Hypertext Transfer Protocol
IMS	Intermediate Mapping Service
INT	Intermediate network
IOIs	Items of interest
IP	Internet Protocol
IPsec	Internet Protocol Security
IPv4	Internet Protocol Version 4
IPv6	Internet Protocol Version 6
ISP	Internet Service Provider

LAP	Level Attachment Point
LLDP	Link Layer Discovery Protocol
LMS	Local Mapping System
Loc/ID Split	Locator/Identifier Split
MAC	Media Access Control
MPE	Mapping Pointer Entry
MPLS	Multiprotocol Label Switching
mRTE	Masked Routing Table Entry
mRUE	Masked Routing Update Entry
MTE	Mapping Table Entry
MTU	Maximum Transmission Unit
NAC	Network Address Confidentiality
NIC	Network Identifier Confidentiality
NLC	Network Locator Confidentiality
OLV	Offset Length Value
OP	Onion Proxy
OR	Onion Router
OWD	One Way Delay
PEKS	Public key Encryption with Keyword Search
PKI	Public Key Infrastructure
QoS	Quality of Service
RIP	Routing Information Protocol
RSA	Ron Rivest, Adi Shamir, and Leonard Adleman
RTT	Round Trip Time
SDN	Software-Defined Networking
SSH	Secure Shell
TCP	Transmission Control Protocol
TLS	Transport Layer Security
TLV	Type Length Value
ToMaTo	Topology Management Tool
UDP	User Datagram Protocol
UML	Unified Modeling Language
VLAN	Virtual Local Area Network
VPN	Virtual Private Network

Chapter 1

Introduction

The current network architecture of the IPv4/IPv6 Internet [Pos81a], [DH98] has been designed in a way that it can provide connectivity and interoperability between a large number of heterogeneous end systems by deploying a single global network layer. However, this principle of a fixed network layer restricts the innovative development and integration of additional in-network services such as *Quality of Service (QoS)*, *Deep Packet Inspection (DPI)*, and *traffic engineering* [Wol10]. Meanwhile, there exist a wealth of approaches for a Future Network Architecture (FNA). Although these approaches differ in their orientation, they all suggest that a network should be service-oriented and flexibly orchestrated from atomic smart in-network services [HSK10].

In these approaches, in-network services orchestrated in a network require certain control data to be signalled for their operability. Here, we can consider various kinds of control data signalling established so far. For example, control data can be put in front of the user data such as packet addresses. Additionally, control data can be signalled on a separate plane as is the case with routing information exchange. Furthermore, control data can be also signalled by coding it into user data such as transcoding multimedia data in the network [VLBA12]. Thus, we can state that in-network services require access to various control data signalled in different ways to utilise the complete functionality of the orchestrated network. Moreover, the diversity and amount of control data required by in-network services rises progressively. Hence, the communication endpoints have to allow more and more access to information about themselves in order to utilise the complete functionality of an orchestrated network. However, the in-network services are able to sniff information. Furthermore, even third parties can sniff information while it is transferred in cleartext in order to operate the in-network services.

Beside these considerations for a FNA, providing information confidentiality for two communicating endpoints is one of the services that has to be provided on the end systems

as well as to be ensured in the network. The de facto method applied so far is end-to-end encryption of information transferred between two endpoints. Cryptographic algorithms, e.g., AES [DR13] and RSA [RSA78] are used in secure communication protocols such as SSH [YL06], TLS [Die08], and IPsec [KS05]. However, in-network services have then no longer access to the encrypted control data and they cannot accomplish their tasks anymore. Thus, utilising the benefits of FNA approaches is not possible anymore, if the communicating endpoints perform end-to-end encryption. Furthermore, not all control data can be transferred in end-to-end encrypted form. For example, the ultimate source and destination addresses of IP packets can be transferred in encrypted form only between IPsec peers (security gateways) by means of IP Encapsulating Security Payload (ESP) protocol in tunnel mode [Ken05].

The issues identified above reveal that realising only the one of the two goals is possible, but together they contradict each other. Thus, we can define the present unsatisfactory state "***either** establishing smart in-network services **or** providing information confidentiality*". This thesis presents the *Blind Packet Forwarding (BPF)* which can represent the beginning of a new state "*establishing smart in-network services **as well as** providing information confidentiality*".

1.1 Motivation

Address-based packet forwarding is the fundamental in-network service present in all connectionless packet-based network architectures except the content-based networks. Basically, for an incoming packet, a network node has the task to forward the packet to one of its neighbour nodes based on the destination address of the packet and the metric used in that network. For selection of the most fitting neighbour node, a network node maintains a routing table which is set up and updated by means of a routing algorithm. In a local network, the destination address of the packet is resolved to a MAC address, and the packet is forwarded to the MAC address of the destination endpoint. Thus, the packet forwarding in its basic form is associated with two in-network services, namely routing and address resolution.

To realise the packet forwarding, a packet has to be addressed in a way that the source and destination endpoint of the packet can be uniquely identified on the network level. In addition, unrestricted access to the network addresses in the packet has to be provided to the network nodes in order to forward the packet in the correct direction. Thus, the network nodes inevitably become authorised entities accessing packet addresses for establishment of the packet forwarding service. Consequently, the network nodes taking part in the transfer

have the possibility to identify which endpoints communicate with each other. Moreover, even third parties, e.g., eavesdroppers on the network nodes, can sniff the packet addresses in order to disclose the communicating endpoints.

We tackle this problem by defining a confidentiality for the network addresses of the packets transferred between two communicating endpoints, where only both endpoints are the authorised entities to access the packet addresses in cleartext. We call this information security property *end-to-end network address confidentiality*, or shortly, *network address confidentiality (NAC)* classifying also network nodes as adversary. BPF is a clean-slate security approach aiming to simultaneously establish the packet forwarding service and to provide NAC. BPF allows to correctly match masked packet addresses with masked routing table entries. In this way confidentiality for packet addresses is provided during transmission as well as during processing by network nodes.

1.2 Organisation of the thesis

After sketching approaches leveraged by this thesis and discussing works related to our approach in Chapter 2, Chapter 3 presents a basic BPF construction that redesigns the packet forwarding and its associated services to blind ones which can still correctly process masked network addresses being based on a simple structure. In addition, this chapter discusses BPF's security benefits and the effects of applying BPF in the current Internet architecture. Moreover, we present BPF's prototype implementation utilising OpenFlow [Ope16a] and its deployment in a real hardware testbed. Finally, this chapter evaluates the implementation of the basic BPF design.

While demonstrating the feasibility to simultaneously establish the packet forwarding service and to provide NAC, the basic BPF design is not suitable to be deployed o a broad scale in the current Internet architecture. Moreover, the basic BPF design introduces a considerable amount of overhead. In Chapter 4, we define requirements to be fulfilled by a suitable architecture for BPF and extend the basic BPF design to two further BPF architectures by means of the existing FNA approaches. Each of both BPF extensions introduces two blindness modes with tolerable overheads achieved by reinterpreting the features of OpenFlow in its implementation. Both BPF designs are deployed in a real hardware testbed to demonstrate their feasibility for practical deployment. At last, we discuss the evaluations for the implementations of both BPF extensions.

While both BPF extensions demonstrate that BPF can in principle be realised by using

the existing FNA approaches, none of the BPF extensions alone can achieve all of the properties required for an adequate BPF design. Chapter 5 combines the beneficial aspects of both BPF extensions into a new design. This new design does not only fulfil the requirements for an adequate BPF design but also introduces a fine-grained, flexible and dynamic blindness providing multiple NAC levels. Moreover, we adapt OpenFlow in order to achieve a BPF implementation which provides high performance and can thus support multiple real-time media communications each with a high sending rate.

Chapter 6 concludes this thesis and gives an outlook on future work. The main results of this thesis have been published in [SBJR13], [SJBR14], and [SJR15].

Chapter 2

Basics

In this chapter, we briefly sketch approaches leveraged by this thesis and discuss works related to our approach.

2.1 Public key Encryption with Keyword Search

The *Public key Encryption with Keyword Search (PEKS)* [BDCOP04] is a cryptographic algorithm enabling to correctly determine for two PEKS ciphertexts whether their cleartext values are the same, without decrypting the ciphertexts. By means of PEKS, we encrypt network packet addresses in order to provide a new information security property called *end-to-end network address confidentiality* which we have introduced in Section 1.1.

In [HYH13], Hsu *et al.* have presented a study of PEKS and its extensions [BSNS08], [Kha06], [RPSL10], [ZCM+12], and [YXZ11]. In this thesis, we apply the PEKS construction based on a bilinear map of elliptic curves [BF01]. It uses two groups G_1 and G_2 of prime order p as well as a bilinear map $e : G_1 \times G_1 \rightarrow G_2$ between them. The map satisfies the following properties:

- Given $g, h \in G_1$ there is a polynomial time algorithm to compute $e(g, h) \in G_2$.
- For any integers $x, y \in [1, p]$, $e(g^x, g^y) = e(g, g)^{xy}$.
- If g is a generator of G_1 then $e(g, g)$ is a generator of G_2.

There are also two hash functions $H_1 : \{0,1\}^* \rightarrow G_1$ and $H_2 : G_2 \rightarrow \{0,1\}^{log\ p}$. Thus, the PEKS algorithm consists of the following methods:

- ***KeyGen(s)***: The security parameter s determines the size p of the groups G_1 and G_2. The function picks a random $\alpha \in \mathbb{Z}_p^*$ and a generator g of G_1. It outputs a public key $A_{pub} = [g, h = g^\alpha]$ and a private key $A_{priv} = \alpha$.

- $\boldsymbol{E(W) = PEKS(A_{pub}, W)}$: It computes $t = e(H_1(W), h^r) \in G_2$ for a random $r \in \mathbb{Z}_p^*$ and outputs a searchable encryption $E(W) = [g^r, H_2(t)]$.
- $\boldsymbol{T(W) = Trapdoor(A_{priv}, W)}$: It outputs an encrypted trapdoor $T(W) = H_1(W)^\alpha \in G_1$.
- $\boldsymbol{Test(E(W), T(V))}$: Let $E(W) = [A, B]$. If $H_2(e(T(V), A)) = B$ then it outputs 1 ($W = V$) otherwise it outputs 0 ($W \neq V$).

In [BDCOP04], Boneh *et al.* have proved that the PEKS construction is semantic-secure, i.e. $E(W)$ and $T(W)$ do not reveal any information about W. The PEKS construction does not allow to decrypt ciphertexts, i.e. the encryption is not invertible. Moreover, the encryption function of PEKS is not deterministic. This means that the function outputs different ciphertexts for each encryption of the same cleartext with the same public key. In contrast to the encryption function of PEKS, its trapdoor function is deterministic. Thus, the function always outputs the same trapdoor value for a pair of a cleartext and a private key. Furthermore, $Test(E(W), T(W))$ outputs 1, if and only if $E(W)$ and $T(W)$ are generated with the same key pair (A_{pub}, A_{priv}).

PEKS is applied in various approaches for different purposes. On the basis of PEKS, Aviv *et al.* have proposed a system for searchable remote encrypted Email storage [ALPK07]. In addition, Kim *et al.* have presented a mechanism supporting secure validation of routing information in the inter-domain routing protocol of the Internet [KXNP08]. Moreover, Liu *et al.* have proposed a privacy preserving keyword search scheme in cloud computing [LWW09]. Furthermore, Shikfa *et al.* have focused on the privacy and confidentiality in context-based and epidemic forwarding [SÖM10]. The last work is most relevant to ours, since it enables to blindly process control data in networks, i.e. to blindly compare message profiles with node profiles. In this work, message profiles are encrypted with the public key of a third party and also trapdoor values for node profiles are generated with its private key. However, this third party acts then as a further in-network service, which can attain knowledge of the identities of message sources and destinations and node profiles.

2.2 Locator/Identifier Split

While an IP address identifies an endpoint (identifier) and describes its network attachment point (locator) at the same time, the *Locator/Identifier (Loc/ID) Split* principle [MZF07] separates the locator functionality from the identifier. Thus, the network address of an endpoint consists of a Loc and an ID part. The ID part serves to locate the endpoint within a local network to which the endpoint is connected, while the Loc part specifies the location of the local network in the entire infrastructure, e.g., in the Internet. Moreover, a mapping system is

needed to map the actual Loc of an endpoint to its ID. Furthermore, the ID of an endpoint has to be resolved to its actual Loc before sending a network packet to the endpoint. The Loc/ID Split principle is regarded as the de facto addressing standard for Future Network Architecture (FNA) and supports scalability, mobility and multihoming by design [Li11], [SG 12].

Currently, there exists a wealth of approaches relying on the Loc/ID Split principle. As in [HMH13], these approaches can be categorised into two classes on the basis of how the ID of an endpoint is resolved to its actual Loc. Approaches (e.g., [AB12], [FCM+09], [MN06], and [PPJB08]) from the first class perform the mapping lookup in endpoints, while intermediate nodes query the mapping system for a mapping information in approaches (e.g., [FFML13], [MHK13], [HSKE09], and [XJ09]) from the second class. Towards achieving an adequate design for Blind Packet Forwarding, we leverage pioneer approaches from each class, which we sketch below.

2.2.1 Global Locator, Local Locator, and Identifier Split

The *Global Locator, Local Locator, and Identifier Split (GLI-Split)* [MHK13] is one of the FNA approaches relying on the Loc/ID Split principle. This approach splits the functionality of IP addresses into global and local locators as well as identifiers. In GLI-Split, identifiers and locators are encoded in regular IPv6 addresses. In this way, a new routing protocol is not needed, and GLI-Split is backward-compatible with the IPv6 Internet. The architecture of GLI-Split divides the network into a global domain (IPv6 backbone) and multiple local domains connected to each other via the global domain. Gateways (border routers of local domains) and nodes of local domains have a local locator that describes their positions within their local domains. Within the global domain, the position of each gateway is described by a global locator. GLI-Split separates global and local routing from each other and performs core routing on global locators and edge routing on local locators. Thus, the changes in a local domain do not affect the global domain, and vice versa.

2.2.1.1 Addressing

GLI-Split defines three types of addresses: identifier address, local address, and global address. The identifier address of an endpoint represents its identifier and does not contain any locator information. This address type is used in the transport layer. In this way, the transport layer sees the same address regardless of the current location of the endpoint. On the network layer, the identifier address is rewritten in an appropriate locator address of the endpoint, and vice versa. This process is called *vertical address conversion* (see Figure 2.1). The local address of the endpoint contains its identifier and the local locator of the edge node to which the endpoint is currently connected. The local address is used for forwarding

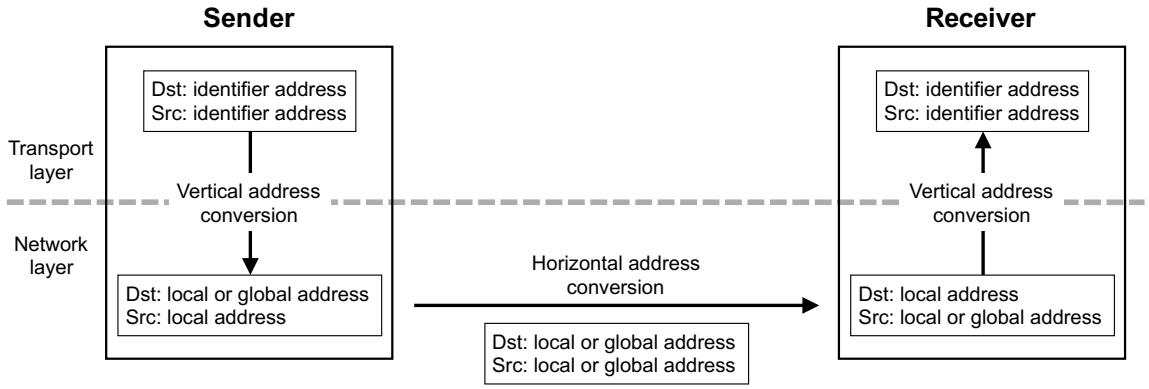


Figure 2.1: Address translations in GLI-Split [MHK13].

within the endpoint's local domain. Gateways at the border of a local domain rewrite local addresses in global addresses, and vice versa. This address translation is called *horizontal address conversion*. A global address of the endpoint contains its identifier and the global locator of a gateway responsible for the local domain. Thus, global addresses are utilised for forwarding within the global domain.

The three address types are encoded by reusing the 128-bit IPv6 address format. A n-bit prefix (GLI-prefix) differentiates GLI-addresses from classical IPv6 addresses. The 64 lower-order bits contain an identifier, while the remaining bits are used for the locator information. The locator part of identifier addresses is filled with padding zeros. In a locator address, two bits following the GLI-prefix determine whether the locator part contains a global or local locator. Global locators are IPv6 prefixes that are globally allocated to gateways from ISPs in the same manner as in the current IPv6 Internet. Local locators are locally assigned according to the topology and management of local domains.

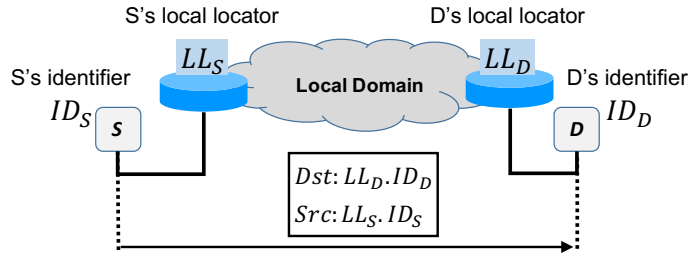
2.2.1.2 Mapping

The mapping system in GLI-Split consists of a global mapping system and a local mapping system in each local domain. The global mapping system maps any identifier to a set of global addresses. The set contains multiple addresses, if the associated local domain is multihomed. For identifiers of endpoints being located in a local domain, the associated local mapping system maintains their mappings to local addresses. In [MHH10], Menth *et al.* have presented how the mapping system works in detail.

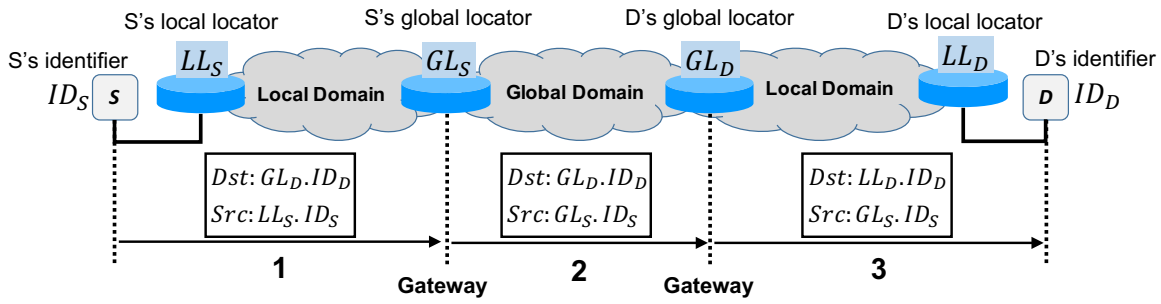
An endpoint newly connected to a local network queries an enhanced DHCP server which responds with a local locator, a set of global locators (if the local domain is multihomed),

and reachability information for the mapping system. After that, the endpoint registers its global and local addresses at the global and local mapping systems. In case of reattachment to another local network, the endpoint repeats this procedure.

Before sending a network packet, the source endpoint S has to resolve the actual locator of the destination endpoint D. For that, the source endpoint queries the local mapping system responsible for the local domain in which the source endpoint is located. If both endpoints reside in the same local domain, the local mapping system responds with a local address of the destination endpoint. Otherwise, the local mapping system forwards the request to the global mapping system which responds with a set of global addresses for the destination endpoint. The set contains multiple global addresses, if the destination local domain is multihomed. The source endpoint can cache the addresses for the following packets so that it does not have to perform a mapping lookup for each packet.



(a) Intra-domain communication.



(b) Inter-domain communication.

Figure 2.2: Packet delivery in GLI-Split [MHK13].

2.2.1.3 Packet delivery

The endpoint S wants to send a packet to the endpoint D. We assume that both endpoints have already registered their global and local addresses at the global mapping system and at the associated local mapping systems. Moreover, it is assumed that the initiating endpoint S

has already found the identifier of the destination endpoint, e.g., via DNS. The source endpoint performs a mapping lookup for D's identifier.

If the source endpoint gets a local address, both endpoints reside in the same local domain (see Figure 2.2a). In this case, the endpoint S uses the local address as the destination address and its own local address as the source address. The packet is locally forwarded to the endpoint D on the basis of the local locator in the destination address.

In case of inter-domain communication (see Figure 2.2b), the endpoint S gets a set of global addresses. If the destination local domain is multihomed, the set consists of multiple addresses. The endpoint chooses one of the global addresses as the destination address and its own local address as the source address. The packet is forwarded in three steps:

1. The packet is forwarded to a gateway of the source local domain by using the default route.

2. Upon receiving the packet, the default gateway replaces the source local address with S's global address which contains the source identifier and the global locator of the gateway. After that, the packet is globally forwarded to the destination local domain on the basis of the global locator in the destination address.

3. When the packet arrives at a gateway of the destination local domain, the gateway queries the local mapping system for D's local address and replaces the destination global address with D's local address. After that, the packet is locally forwarded to the endpoint D on the basis of the local locator in the destination address.

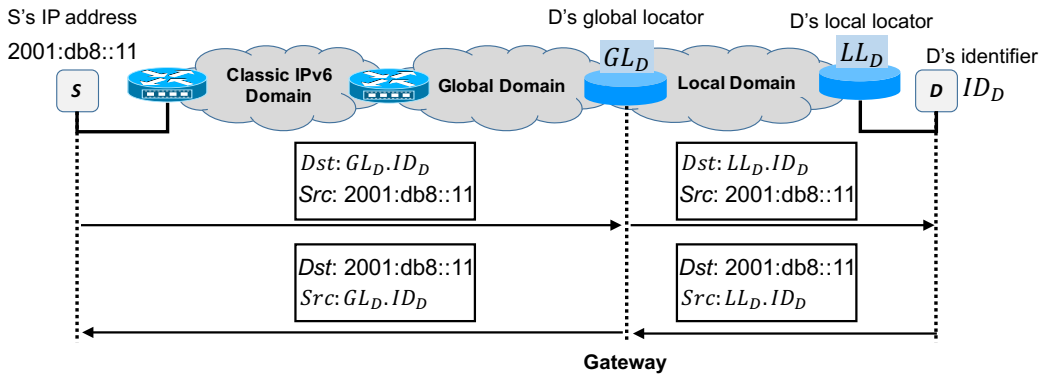


Figure 2.3: Communication between an IPv6 host and a GLI-Split host [MHK13].

If the endpoint S resides in a classical IPv6 domain (see Figure 2.3), i.e. the endpoint is a classical IPv6 host, the source endpoint obtains D's global address encoded in IPv6 address

format directly from DNS. The endpoint S uses the global address as the destination address and its own regular IPv6 address as the source address. According to the global locator in the destination address, the packet is forwarded to a gateway of the destination local domain. From then on, it is proceeded as described in step 3 above. In the reverse case, steps 1 and 2 are performed. After that, the gateway of the source local domain forwards the packet on the basis of the conventional IPv6 address in the destination field.

2.2.2 Hierarchical Architecture for Internet Routing

The *Hierarchical Architecture for Internet Routing (HAIR)* [FCM+09] is a FNA approach which relies on the Loc/ID Split principle. This approach defines a n-level-based hierarchical scheme which reflects the current Internet structure (see Figure 2.4). There, local networks placed at level n are called *Edges*. At levels $n-1$ to 2 we have the so called *Intermediate (INT)* networks which consist only of routers and ensure the reachability between the attached Edges and INTs of the next higher level. We can conceive the INTs as access providers or enterprise networks. The top level of the hierarchy is the so called *Core* providing routing reachability between the INTs at level 2. Each router in the Core belongs to one administrative domain and is under the control of a single ISP, just like in the today's Internet backbone. The organisation of hierarchy levels is not restricted to be globally symmetrical. Domain providers can independently organise their own domains in various hierarchy levels. Each node in a domain has a locator describing its position within the domain.

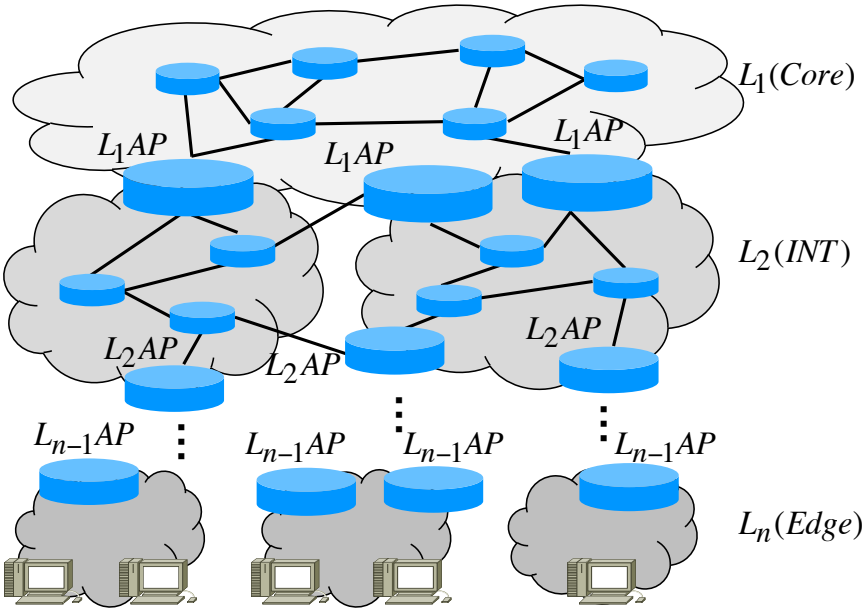


Figure 2.4: HAIR's n-level-based hierarchical scheme [FCM+09].

Levels 1 to n are connected via so called *Level Attachment Points (LAPs)* acting as gateway nodes. A routing domain at level k is connected to a routing domain at level $k+1$ via L_kAP. A LAP has at least two locators which describe its positions within the domains connected with each by the LAP. In HAIR, the routing is performed domain-wise so that a network node in a routing domain at level k maintains routing entries only for the network nodes in the same domain at level k, $1 \leq k \leq n-1$. In this way, the changes in a domain do not affect the neighbouring domains.

2.2.2.1 Addressing

The address of an endpoint consists of its identifier and actual locator sequence. It is assumed that identifiers are organised in a global flat namespace. The locator sequence of an endpoint defines the LAPs to be traversed for forwarding a packet from the Core to the endpoint, which is similar to loose source routing. The address of the endpoint X with the identifier ID_X is

$$Addr_X : \underbrace{L_1AP_X|...|L_{n-1}AP_X}_{Loc_X\text{: locator sequence}} \mid ID_X.$$

Here, $L_1AP_X, ..., L_{n-1}AP_X$ are the locators of LAPs that have to be traversed to reach the endpoint X. The length of a locator sequence can be variable. In the prototype implementation of HAIR, IPv6 addresses are used for both identifiers and locators.

2.2.2.2 Mapping

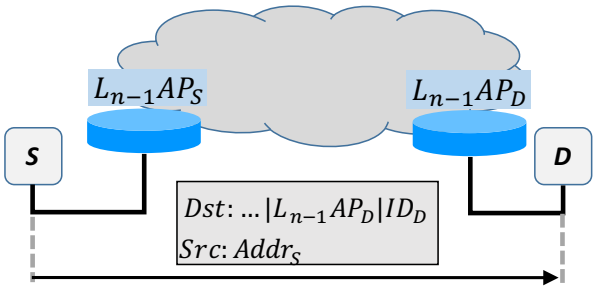
HAIR defines a hierarchical mapping system which binds the actual locator sequence to the identifier of an endpoint. The mapping system consists of a *Core Mapping Service (CMS)* and multiple *INT Mapping Services (IMSs)*. An IMS holds the actual mappings for the endpoints belonging to that INT, and the CMS keeps a pointer to the IMS maintaining the current mapping for each identifier. The CMS is provided by the Core, while $IMSs$ are administered by INTs. Although HAIR does not specify how to implement the mapping service, it proposes to use any distributed directory service (e.g., DNS) for the CMS and $IMSs$.

HAIR does not specify how to perform a mapping registration and lookup. In the prototype implementation of HAIR, the mapping system is realised as a single HTTP server which keeps mappings. Moreover, an endpoint gets the locators of the associated LAPs from a DHCP server and registers the mapping of its identifier to the composed locator sequence at the mapping server via HTTP. Furthermore, the identifier of an endpoint is resolved to its actual locator sequence by means of an HTTP-like query to the mapping server.

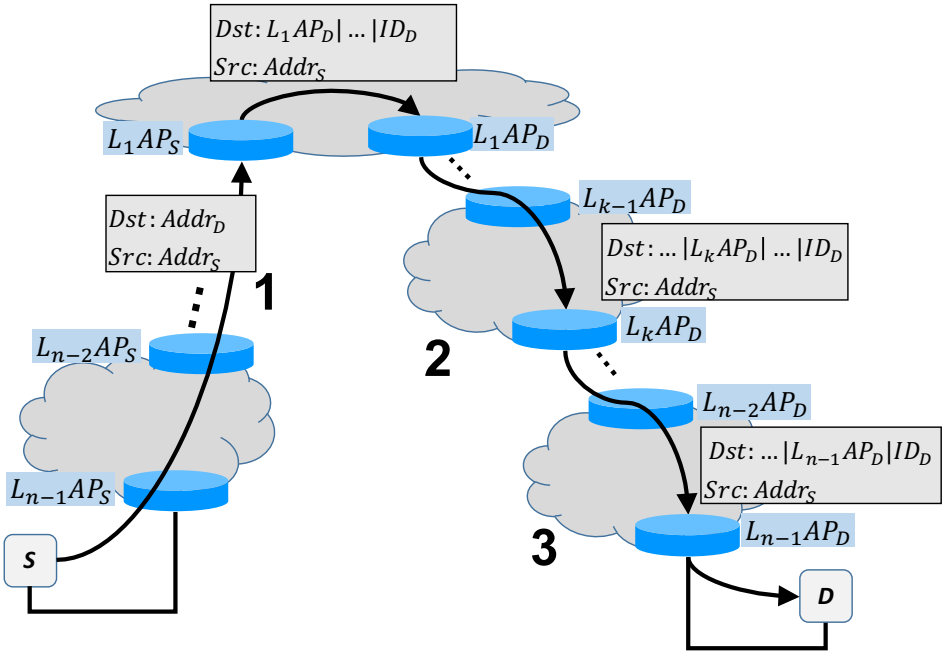
2.2.2.3 Packet delivery

The endpoint S with the identifier ID_S wants to send a packet to the endpoint D. It is assumed that both endpoints have already registered their mappings at the mapping system, and S has already found D's identifier ID_D, e.g., via DNS. First, the endpoint S resolves D's identifier to its actual locator sequence by querying the mapping system.

If both endpoints reside in the same INT (see Figure 2.5a), the packet is locally forwarded to the destination Edge on the basis of the last locator in the destination locator sequence. Eventually, the packet arrives at the destination edge node which resolves the destination identifier to a MAC address. Finally, the packet is forwarded to the MAC address.



(a) Intra-domain communication.



(b) Inter-domain communication.

Figure 2.5: Packet delivery in HAIR [FCM+09].

If the endpoints S and D are located in different INTs, the endpoint S gets the locator sequence $Loc_D : L_1AP_D|...|L_{n-1}AP_D$ from the mapping system. Thus, the destination address of the packet is $Addr_D : Loc_D \mid ID_D$, and its source address is $Addr_S : Loc_S \mid ID_S$. The packet is forwarded in three steps (see Figure 2.5b):

1. The packet is forwarded up to the Core either by using the default route or by utilising the locator sequence Loc_S in the source address.
2. The packet is forwarded domain-wise. Here, L_kAP_D is taken as the basis locator for packet forwarding in a routing domain at level k, $1 \leq k \leq n-1$.
3. Eventually, the packet arrives at the destination Edge. The edge node resolves D's identifier to its MAC address and forwards the packet to the MAC address.

2.3 OpenFlow

Software-Defined Networking (SDN) [Fun12] is a network architecture which proposes decoupling the control plane and the forwarding plane of network devices (e.g., switches and routers) from each other and centralising the control logic in a software-based controller (see Figure 2.6). While the SDN controller maintains the network intelligence, the network devices perform instructions from the controller. *OpenFlow* [Ope16a] is a SDN protocol which specifies the components and the basic functions of network devices as well as their management by a controller.

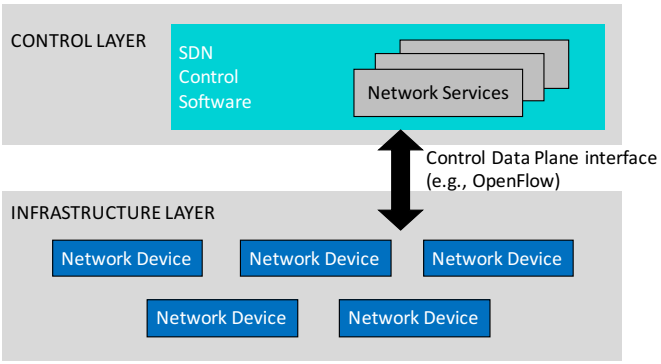


Figure 2.6: SDN Architecture [Fun12].

Due to the SDN approach, OpenFlow offers support for innovative networking concepts, such as Blind Packet Forwarding, in real hardware testbeds without a lot of effort. In this way, new networking approaches can be evaluated in realistic environments. Beside this aspect

of networking experiments, the management of experimental environments is also crucial. The *topology management tool (ToMaTo)* [MSC14] developed as part of the German-Lab project [SRZ+14] allows to provide as well as manage realistic and lightweight components with regard to high realism and parallelism by using multiple virtualisation technologies.

The OpenFlow specification is under continuous development, and it is available in different versions [Ope16b]. A detailed list of all version changes can be found in [Opeb]. Moreover, the currently available deployments of OpenFlow and the associated controllers are given in [Ope16c]. For this thesis, we leverage OpenFlow version 1.3 [Opea]. An OpenFlow-enabled network device called *OpenFlow switch* consists of an *OpenFlow datapath* and an *OpenFlow channel* (see Figure 2.7), which we discuss below on the basis of OpenFlow 1.3.

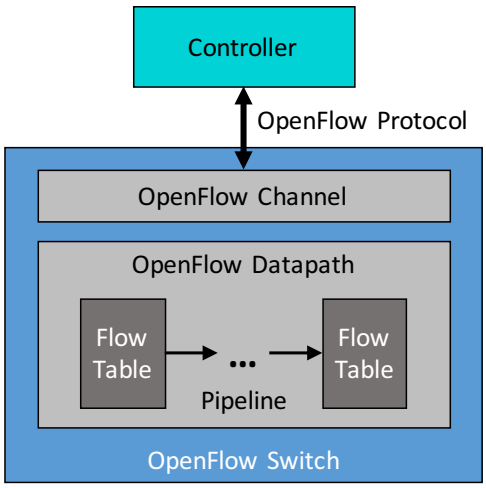


Figure 2.7: OpenFlow switch [Opea].

2.3.1 OpenFlow datapath

An OpenFlow datapath consists of one or more *flow tables*. A flow contains one or multiple match fields, a priority for matching precedence, and a set of instructions. A flow match field is described using the *Type Length Value (TLV)* format. Together the type and the length in a flow match field define a packet field (e.g., Ethernet destination address (six bytes)) whose value is matched against the value in the match field. The match fields of a flow are logically linked to each other by means of the AND operator. Thus, if the values of the packet fields specified by the flow match fields fit the values in the match fields, it is said that the flow matches the packet, or vice versa. The types supported by OpenFlow 1.3 are listed in [Opea]. A flow match field can also define the ingress port or the metadata which is a register to carry information from a flow table to the next one. A flow is identified by its match fields and priority.

Upon receiving a packet, the datapath parses the packet on the basis of pre-defined types defining the respective fields of the packet according to supported protocols (see [Opea] for protocols supported by OpenFlow 1.3). Next, the packet (i.e., its field values) goes through a pipeline (flow tables). Here, flows matching the packet are determined in priority order as well as the instructions defined in these flows are performed on the packet. An instruction can modify pipeline processing such as directing the packet to another flow table or contains actions which are performed either immediately or after exiting the processing pipeline.

An action of a flow can forward the matching packet via a specified port (*Output*), send the packet to the controller (*Controller-Output*), and drop the packet. Moreover, an action can push/pop certain tags (e.g., VLAN, MPLS) and modify values of respective packet fields specified by their types and lengths. A detailed list of possible actions as well as supported tags and packet fields in OpenFlow 1.3 can be found in [Opea].

2.3.2 OpenFlow channel

An OpenFlow datapath is connected to a controller by means of the OpenFlow channel through which the controller manages the datapath and receives events from the datapath. Here, the datapath initiates a standard TLS or TCP [Pos81b] connection to the controller, and the connection is identified by the OpenFlow-specific ID of the datapath. The OpenFlow protocol specifies the format of all OpenFlow channel messages to be classified into three types:

- *Controller-to-datapath* messages are initiated by the controller. By means of such messages, the controller can request datapath capabilities (*Features*), set/query configuration parameters (*Configuration*), add/modify/delete flows (*Modify-State*), and send packets out of a specified port on the datapath (*Packet-Out*).
- By means of *asynchronous* messages, a datapath can inform the controller of packet arrival and state changes (e.g., flow removal or port configuration change). If a flow instructs sending a matching packet to the controller, the datapath can send the packet to the controller by means of a *Packet-In* message. This message contains the OpenFlow-specific ID of the datapath and the ingress port of the packet as well as the entire packet or a specific part of the packet, e.g., only the Ethernet and IP headers. In the last case, the remaining part of the packet is cached at the datapath, and the message also contains the cache ID for the packet.
- Between a datapath and controller, *symmetric* messages are exchanged upon connection setup (*Hello*) and for verifying the liveness of the connection (*Echo request/reply*).

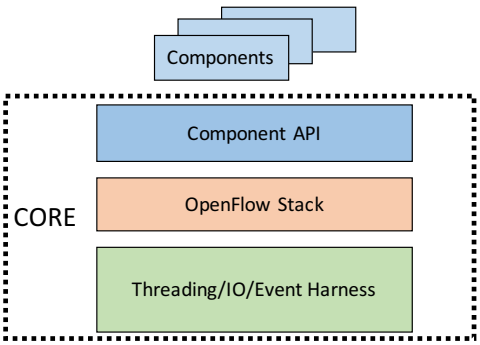


Figure 2.8: NOX architecture [nox16].

2.3.3 NOX

NOX [GKP+08] is an OpenFlow controller written in C/C++. NOX provides a high level programmatic interface for management and development of network services encapsulated by *components* (e.g., switching, routing). The architecture of NOX is given in Figure 2.8. All components inherit from the class *Component*. By means of this class, components can send controller-to-datapath messages and become listener of events generated by asynchronous messages. The most important events are

- *Datapath-Join*: Resolved when a new datapath is detected.
- *Datapath-Leave*: Resolved when a datapath leaves the network.
- *Packet-In*: Resolved when a packet is received from a datapath.
- *Datapath-Port-Description*: Resolved when port description of a datapath is received.

After detecting a new OpenFlow datapath, a NOX component queries the datapath to send its port description. After that, the component defines a flow whose match field is wildcarded. This flow matches any packet and has the lowest priority as well as instructs to send matching packets to the controller. Thus, the flow matches a packet, if the packet has not matched the other flows. In this regard, the first packet of a new traffic seen by a datapath is sent to the controller. The component handles the packet, instructs the datapath to set one or more correspondent flows, and sends the packet out of a specified port on the datapath. The remaining packets of the traffic are then forwarded on the basis of the flows.

2.4 Anonymous communication

Confidentiality of information, which can be used to identify a subject, can deduce a certain degree of subject anonymity against adversaries. In Section 2.4.1, we first discuss the

anonymity properties defined by [PH05]. Anonymity systems can be classified in high- and low-latency anonymity systems as is the case in [EY09]. High-latency anonymity systems (e.g., [Cha81], [SDS02], and [Dan03]) are designed to be applied for non-interactive applications tolerating delays of several hours or more, such as email. Since our approach aims to support transparency to applications, high-latency anonymity systems fall out of scope of our discussion. In [RW10], Ren *et al.* divide low-latency anonymity systems into the following categories: Mixnet-based schemes, routing-based techniques, peer-to-peer networks. Below, we sketch pioneer approaches from each category. The anonymity properties of these approaches are discussed in Section 2.4.6.

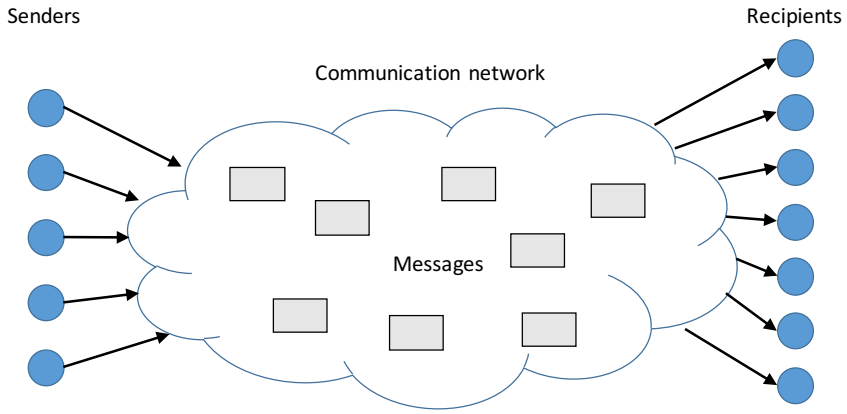


Figure 2.9: Anonymity model [PH05].

2.4.1 Anonymity properties

In [PH05], Pfitzmann *et al.* define the following setting:

- *Senders* send messages to *recipients* using a communication network (see Figure 2.9).
- An *attacker* may be an outsider monitoring communication lines, but also an insider able to participate in normal communications or/and to control at least some stations.
- A *subject* is an acting entity (e.g., sender or recipient) which might cause an action (e.g., sending or receiving).
- An *identity* is any subset of attributes of an entity which distinguishes this entity from all other entities within any set of entities (e.g, a host can be identified on the basis of its IP address).
- *Items of interest (IOIs)* can be subjects, messages, actions, and identities in which the attacker might have interest for its observation.

With regard to the setting above, the following anonymity properties are defined:

- *Anonymity*: The state of being not identifiable within a set of subjects (the anonymity set).
- *Sender/recipient unlinkability*: A particular message is not linkable to any sender/recipient and no message is linkable to a particular sender/recipient.
- *Relationship unlinkability*: It is untraceable who communicates with whom. In other words, sender and recipient are unlinkable to each other.
- *Sender/recipient unobservability*: It is not detectable whether any sender/recipient sends/receives.
- *Relationship unobservability*: It is not detectable whether anything is sent out of a set of could-be senders to a set of could-be recipients. In other words, it is not detectable whether within the relationship unobservability set of all possible sender-recipient(s)-pairs, a message is exchanged in any relationship.

Here, the following relationships are stated between the anonymity properties:

- Sender/recipient unobservability implies sender/recipient unlinkability.
- Relationship unobservability implies relationship unlinkability.
- Sender/recipient unlinkability implies relationship unlinkability.
- Sender/recipient unobservability implies relationship unobservability.

2.4.2 IP Encapsulating Security Payload (ESP) protocol in tunnel mode

ESP protocol in tunnel mode [Ken05], shortly ESP, encrypts an IP packet including its IP header and encapsulates the encrypted packet in a new IP packet with a new IP header containing the IP addresses of IPsec peers (security gateways) connecting the private networks with the public network. Thus, the original packet with its ultimate source and destination addresses are transferred between the security gateways in encrypted form. ESP classifies the private networks including the security gateways as trusted and the public network as untrusted. Hence, security gateways and network nodes within the private networks are authorised entities having access to the ultimate source and destination addresses of the original packet in cleartext. To the best of our knowledge, ESP is the only approach aiming to achieve network address confidentiality primarily.

2.4.3 Tor

Tor [DMS04] is a mixnet-based overlay network aiming to anonymise TCP connections. Tor clients choose a path through the network and build a *circuit* by negotiating a symmetric key with each node called *onion router (OR)* in the path. Traffic is packed in fixed-size *cells* which are encrypted layer-wise with each shared key, like the layers of an onion. The cells flow down the circuit and are unwrapped by a symmetric key at each OR.

In the Tor network, each OR maintains a TLS connection to each other OR. Tor users run *onion proxies (OPs)* communicating with ORs using TLS connections. By means of *directory servers* (redundant, well-known ORs) acting as HTTP servers, OPs can fetch current network state and *descriptors* of ORs (their keys, addresses, bandwidths, and so on), and other ORs can upload their descriptors. Each OR keeps a long-term identity key and a short-term onion key. An OR uses its identity key to sign TLS certificates and its descriptor. Moreover, the OR decrypts circuit setup requests from OPs by means of its onion key.

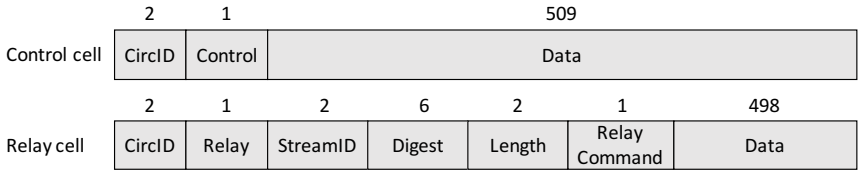


Figure 2.10: Cell structures in Tor [DMS04].

2.4.3.1 Cell

Traffic between two ORs as well as between an OP and OR is packed in fixed-size cells (512 bytes) and transferred over the associated TLS connection. A cell consists of a header and a payload (see Figure 2.10). The header contains a circuit identifier (circID) and a command. On the basis of the command, a cell is either a *control cell* used to set up/destroy a circuit or a *relay cell* carrying end-to-end stream data. The control cell commands are *create*, *created*, and *destroy*. A relay cell includes an additional header called *relay header*. The relay header contains a stream identifier (streamID), an end-to-end integrity checksum (*digest*), the relay payload length, and a relay command. The relay commands are

- *relay extend* for extending the circuit by a hop,
- *relay extended* to acknowledge the circuit extension,
- *relay begin* inducing to open a TCP connection,

- *relay connected* to notify the OP of the TCP connection setup,
- *relay data* for transferring TCP data, and
- *relay end* closing a TCP connection.

The relay header and payload are encrypted or decrypted together using the 128-bit AES cipher to generate a cipher stream.

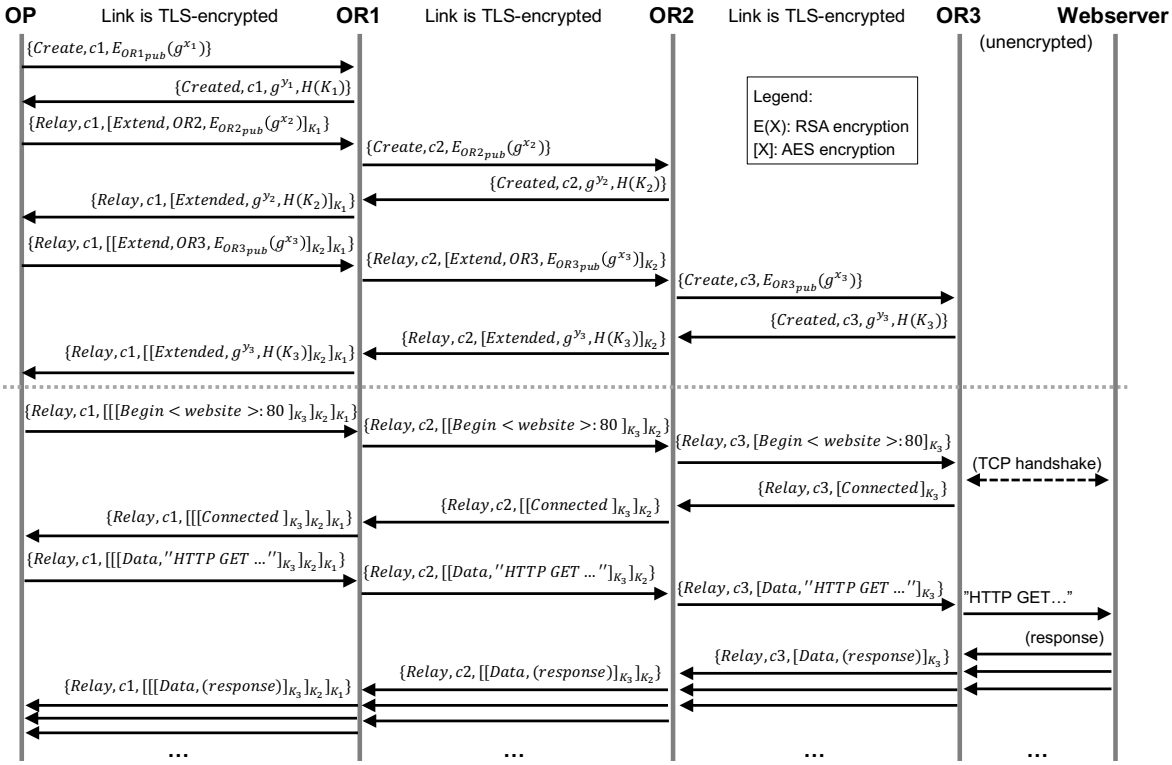


Figure 2.11: Circuit setup and transferring TCP payloads in Tor [DMS04].

2.4.3.2 Circuit

An OP periodically builds a new circuit, rotates between the existing circuits once a minute, and expires the old ones. To build a new circuit, the OP selects a path of usually three ORs ($OR1$, $OR2$, $OR3$) and negotiates a symmetric key with each OR in the path (see Figure 2.11):

1. The OP selects a new circID $c1$, maps this to $OR1$ as the successor, and generates the first half of the Diffie-Hellman handshake (g^{x_1}) encrypted with $OR1$'s onion key by means of RSA. After that, the OP sends a *create* cell containing the circID $c1$ and the encrypted g^{x_1} to $OR1$.

2. Upon receiving the *create* cell, $OR1$ maps the circID $c1$ to the OP as the predecessor. Moreover, $OR1$ decrypts the cell payload content and generates g^{y_1} as well as a cryptographic hash of the negotiated key $K_1 = g^{x_1 y_1}$. After that, $OR1$ packs the second half of the Diffie-Hellman handshake as well as the hash of the key into a *created* cell with the circID $c1$ and sends it back to the OP. Thus, the circuit between the OP and $OR1$ is established, and they share the secret key K_1 mapped to the circID $c1$. By means of the circuit, the OP and $OR1$ can send relay cells encrypted with the symmetric key K_1 to each other.

3. For extending the circuit to $OR2$, the OP uses the circuit to send a *relay extend* cell to $OR1$. This cell has the circID $c1$ and contains $OR2$'s address and g^{x_2} encrypted with $OR2$'s onion key by means of RSA. The relay header and payload of the cell is encrypted with K_1 by means of AES.

4. Upon receiving the *relay extended* cell, $OR1$ looks up the key mapped to the circID $c1$ and decrypts the cell with K_1. $OR1$ selects a new circID $c2$ and maps this to $OR2$ as the successor. Moreover, the circIDs $c1$ and $c2$ are correlated with each other. After that, the encrypted g^{x_2} is packed into a *create* cell which gets $c2$ as its circID. Finally, the *create* cell is sent to $OR2$.

5. For the incoming *create* cell, $OR2$ proceeds in the same manner as $OR1$. Thus, $OR2$ maps the circID $c2$ to $OR1$ as the predecessor and responds with a *created* cell which gets $c2$ as its circID and contains g^{y_2} and a cryptographic hash of the negotiated key $K_2 = g^{x_2 y_2}$ mapped to the circID $c2$.

6. After receiving the *created* cell, $OR1$ packs its payload content into a *relay extended* cell which gets the circID correlated with $c2$, namely $c1$. After that, the cell is sent to the address mapped to the circID $c1$ as the predecessor, namely to OP's address. Thus, the OP and $OR2$ share the secret key K_2 mapped to the circID $c1$ as the second key.

7. To extend the circuit to the last OR, the OP proceeds analogously and negotiates the key K_3 with $OR3$.

Thus, the OP shares a symmetric key with each OR (K_1, K_2, K_3), and each OR on the circuit learns its predecessor OR/OP (IP address) and successor OR (IP address), which are mapped to the circuit IDs that are selected hop-wise.

2.4.3.3 Transferring TCP payloads

For opening a TCP connection to a given address and port, a Tor user requests its OP that chooses the newest circuit or creates a new one if needed. Here, we assume that the OP

selects the circuit created above. The OP then packs the address and port into a *relay begin* cell which gets $c1$ as its circID. The cell payload is encrypted layer-wise with the keys K_3, K_2, and K_1 in this order (see Figure 2.11). After that, the OP sends the cell to the OR mapped to the circID as the successor, namely to $OR1$. For the incoming relay cell from a predecessor, an OR on the circuit looks up the key mapped to the circID of the cell and decrypts the outer layer with this key. After that, the OR finds the successor circID correlated with the circID of the cell, and thus, the next hop for the cell. The OR updates the circID in the cell and sends the cell to the successor OR. In this way, $OR3$ gets the cell payload in cleartext.

After connecting to the specified host, $OR3$ responds with a *relay connected* cell which gets $c3$ as its circID. The cell payload is encrypted with the key mapped to the circID $c3$, namely with K_3. After that, the cell is sent to the OR mapped to $c3$ as the predecessor, namely to $OR2$. For an incoming relay cell from a successor, an OR on the circuit finds the predecessor circID correlated with the circID of the cell, and then, the key mapped to the predecessor circID. The OR encrypts the cell payload with this key and updates the circID in the cell which is sent to the predecessor. After receiving the cell, the OP decrypts the cell payload layer-wise with the keys K_1, K_2, and K_3 in this order. The *relay data* cells flow down and up the circuit in the same manner as the *relay begin* and *relay connected* cells.

2.4.4 Crowds

The Crowds system [RR98] is a routing-based technique for anonymous web browsing. A *crowd* can be considered as a collection of endpoints running special web proxies called *jondos*. When a jondo is started, the jondo registers itself at a server called *blender*. The new jondo gets the membership of the crowd and is reported to other jondos. Each jondo shares an encryption key with each other jondo, and the connection between each two jondos is encrypted. Web requests and replies flow via one or more jondos which are randomly selected hop-wise.

2.4.4.1 Membership

The blender is a central server which controls membership in a crowd and reports it to crowd members. A new user establishes an account with the blender, i.e, an account name and password. If the user starts a jondo, the jondo registers itself at the blender by means of the account name and password. The blender adds the new jondo (i.e., its IP address, port number, and account name) to its member list and reports the list back to the new jondo as well as informs each other jondo of the new jondo. Additionally, the blender generates a key for each other jondo and reports these keys to the new jondo. These keys are encrypted under the account password of the new jondo. Moreover, each of these keys is reported to the

associated jondo and encrypted under the account password of that jondo. Thus, the new jondo shares an encryption key with each other jondo.

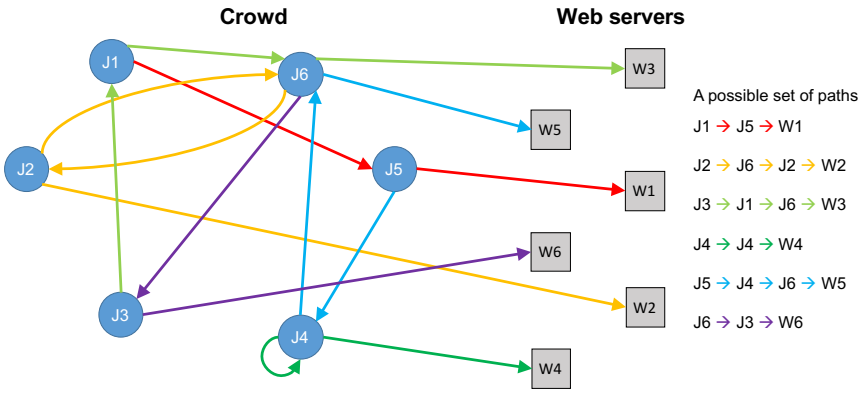


Figure 2.12: Paths in a crowd [RR98].

2.4.4.2 Transferring web requests/replies

A crowd user selects the jondo running on the user's computer as its web proxy. Thus, requests coming from the user's browser are forwarded to the jondo. For the first request to a given web server, the jondo randomly picks a jondo in the crowd, caches the selected jondo as its successor, and forwards the request to the selected jondo. Here, it is also possible that the initiator jondo selects itself. When receiving a request, a jondo either forwards the request to another jondo selected at random, or it submits the request to the web server. The probability for forwarding the request to a further jondo is more than fifty percent. If the jondo decides to forward the request, it caches the jondo from which the request came as its predecessor, and the jondo to which the request will be forwarded as its successor. Subsequent requests from the initiator jondo to the web server take the same path, and server replies travel back the path in reverse. A possible set of such paths is shown in Figure 2.12.

2.4.5 Tarzan

Tarzan [FM02] is a peer-to-peer network aiming to anonymise IP packets. Peers discover each other decentrally and relay packets for each other. Packets are forwarded through pre-built tunnels and encrypted layer-wise with the symmetric keys of peers belonging to the tunnels. Each peer selects a certain number of peers (mimics) with which the peer exchanges dummy traffic (mimic traffic).

2.4.5.1 Peer discovery

In the Tarzan network, a new peer discovers other peers by means of a gossip-based algorithm [HBLL99]. Here, the new peer randomly selects a peer from a set of bootstrap peers. After that, the new peer learns the neighbour peers (their IP addresses, port numbers, and public keys) of the bootstrap peer and reports itself to these peers. The new peer then selects another peer from the new set of known peers at random and repeats the process analogously. After discovering the network, the new peer validates peers which correctly respond to a discovery request including a random nonce.

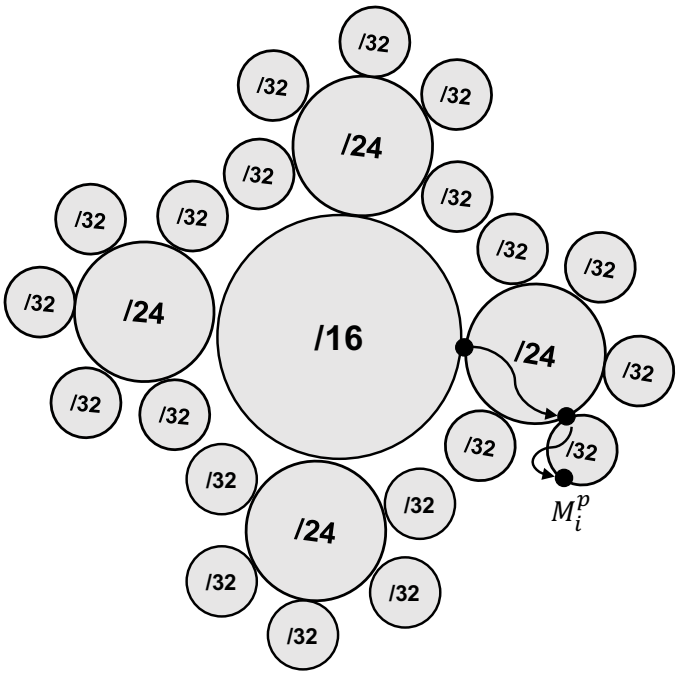


Figure 2.13: Tarzan's three-level hierarchy ring table and mimic selection [FM02].

The new peer maps IP addresses of validated peers into a three-level hierarchy ring table (see Figure 2.13). Here, the ring at the first level keeps hash values of all known /16 domain addresses. Each ring at the second level holds hash values of /24 subnet addresses belonging to one of the 16-bit address spaces from the ring at the first level. Consequently, rings at the third level maintain hash values of peer addresses belonging to each /24 subnet from the second level. For hashing, the current date and the leading d-bits of an address are taken as parameters: $id_d := hash(addr/d, date)$, where $d \in \{16, 24, 32\}$.

The new peer n selects k peers[1] (mimics) with which the peer exchanges dummy traffic

[1]The value k is a global parameter.

(mimic traffic). Each mimic is selected in three rounds (see Figure 2.13). For selecting the ith mimic M_i^n, the peer recursively applies the hash function i times to $(n.addr/16, date)$: $id_{16}^i := hash^i(n.addr/16, date)$. The peer takes the smallest identifier in the ring at the first level, which is bigger than, or equal to, the calculated hash value: $lookup(id_{16}^i) := min\{id \mid id \geq id_{16}^i\}$. In the second round, the peer computes the hash value id_{24}^i and performs $lookup(id_{24}^i)$ in the ring at the second level, which is referenced by $lookup(id_{16}^i)$. In the last round, the peer analogously performs $lookup(id_{32}^i)$ in the associated ring at the third level. Thus, M_i^n is the peer whose hash value is equal to $lookup(id_{32}^i)$ from the third level. In this way, mimics are selected from different domains in order to ensure that they can not be under the control of a single adversary [2].

2.4.5.2 Tunnel setup

To select peers for a tunnel of length l, the initiator peer p randomly chooses a mimic M_i^p from its mimic set $\{M_1^p ... M_k^p\}$. After that, p randomly selects a mimic $M_j^{M_i^p}$ from M_i^p's mimic set $\{M_1^{M_i^p} ... M_k^{M_i^p}\}$. This process is repeated for l peers.

After selecting the peers $\{p_1, ..., p_l\}$, the initiator peer sends an establishment request to each peer p_i. The request contains the forward and backward symmetric keys (fk_{p_i}, bk_{p_i}), the IP addresses of the predecessor and successor peers (p_{i-1}, p_{i+1}), and the pairwise flow identifiers to tag data packets to the forward and backward flows. Moreover, the request is encrypted with the public key of the peer p_i and relayed as a normal data packet from p_1 through p_{i-1}.

2.4.5.3 Packet relaying

Once the tunnel is established, the initiator peer can begin with the transfer of IP data packets from an application. To send a packet, the initiator peer first rewrites the real source address of the packet with a random address and creates the onion $E_{fk_{p_1}}(...E_{fk_{p_l}}(packet)...)$ tagged to the forward flow. After that, the peer encapsulates the onion and its tag in a UDP [Pos80] packet and sends it to p_1. Upon receiving the UDP packet, each peer p_i decrypts the outer layer of the onion with its symmetric key fk_{p_i}, retags the remaining onion, encapsulates the result in a new UDP packet, and sends the packet to the successor peer. Eventually, the packet arrives at the last peer that decrypts the last layer of the onion and gets the IP packet of the initiator peer. Finally, the last peer on the circuit writes its IP address into the source address field of initiator's IP packet and sends it to the destination endpoint. The reply packets take the same tunnel path in reverse, and each peer puts its encryption layer using its backward symmetric key. Thus, the initiator peer gets the onion $E_{bk_{p_1}}(...E_{bk_{p_l}}(packet)...)$.

[2] Tarzan assumes that peers being under the control of an adversary are located in the same IP prefix space.

2.4.6 Analysis

Although ESP does not aim to provide any anonymity property explicitly, ESP provides a restricted sender/recipient and relationship unlinkability within the public network with regard to the ultimate source and destination addresses of packets. However, adversaries within the private networks can link packets to their ultimate source and destination endpoints and know who communicates with whom on the basis of their addresses. In addition, the anonymity properties do not apply to security gateways. Thus, any adversary within the public network can link traffic to the security gateways and detect which security gateways communicate with each other.

Network packets carrying Tor cells from/to an initiator endpoint have its address in cleartext. Thus, the entry OR has access to the network address of the source endpoint in cleartext. The same with regard to the network address of the destination endpoint applies to the exit OR. However, no single OR alone has access to the cleartext network addresses of the source and destination endpoints simultaneously. Nonetheless, this is possible for a global adversary having the ability to sniff the incoming/outgoing network packets to/from the entry/exit OR. But the adversary has to correlate these network packets with each other, which is also possible by means of traffic analysis techniques stated in [DC07]. Thus, sender/recipient unlinkability is not provided by Tor. In contrast to that, Tor aims to supply relationship unlinkability anyway. Moreover, Tor also prevents traffic analysis attacks being based on packet size by means of cells of a fixed size. However, a global adversary can link traffic to communicating endpoint pairs by means of timing analysis attacks [LRWW04].

Recipient unlinkability is not the case in the Crowds system, since each jondo on a path knows which web server is requested. Although the second jondo on the path has access to the cleartext address of the initiator jondo, it does not know with certainty whether the request comes from the initiator jondo or from a jondo that just forwards the request. However, the actual initiator jondo of a request can be disclosed by means of predecessor attacks presented in [WALS04]. Thus, sender unlinkability is also not provided. Due to the ability to link requests to clients and servers, their relationships are also linkable.

The last peer on a path in Tarzan has access to the cleartext address of the destination endpoint. The same applies to the traffic between the initiator peer and the first peer on the circuit. However, it cannot be stated with certainty from which peer the packets originate. Moreover, the real data traffic is masked by means of mimic traffic. Nonetheless, an adversary controlling the first and last peer on the path can disclose the actual initiator peer by means of the intersection attack [WALS02]. Thus, sender/recipient unlinkability is not provided by

Tarzan. Furthermore, an adversary, which controls the first and last peer on the path and performs the intersection attack, is able to link packets to their senders and recipients. Hence, relationship unlinkability is also not supplied by Tarzan.

In the next chapter, we present a basic design for Blind Packet Forwarding (BPF) on the basis of a basic addressing structure. The basic BPF design demonstrates the feasibility to provide end-to-end network address confidentiality which deduces sender/recipient and relationship unlinkabilities.

Chapter 3

A basic design for BPF

Address-based packet forwarding is the fundamental in-network service present in all connectionless packet-based network architectures except the content-based networks. Basically, for an incoming packet, a network node has the task to forward the packet to one of its neighbour nodes based on the destination address of the packet and the metric used in that network. For selection of the most fitting neighbour node, a network node maintains a routing table which is set up and updated by means of a routing algorithm. In a local network, the destination address of the packet is resolved to a MAC address, and the packet is forwarded to the MAC address of the destination endpoint. Thus, the packet forwarding in its basic form is associated with two in-network services, namely routing and address resolution.

To realise the packet forwarding, a packet has to be addressed in a way that the source and destination endpoint of the packet can be uniquely identified on the network level. In addition, unrestricted access to the network addresses in the packet has to be provided to the network nodes in order to forward the packet in the correct direction. Thus, the network nodes inevitably become authorised entities accessing packet addresses for establishment of the packet forwarding service. Consequently, the network nodes taking part in the transfer have the possibility to identify which endpoints communicate with each other. Moreover, even third parties, e.g., eavesdroppers on the network nodes, can sniff the packet addresses in order to disclose the communicating endpoints.

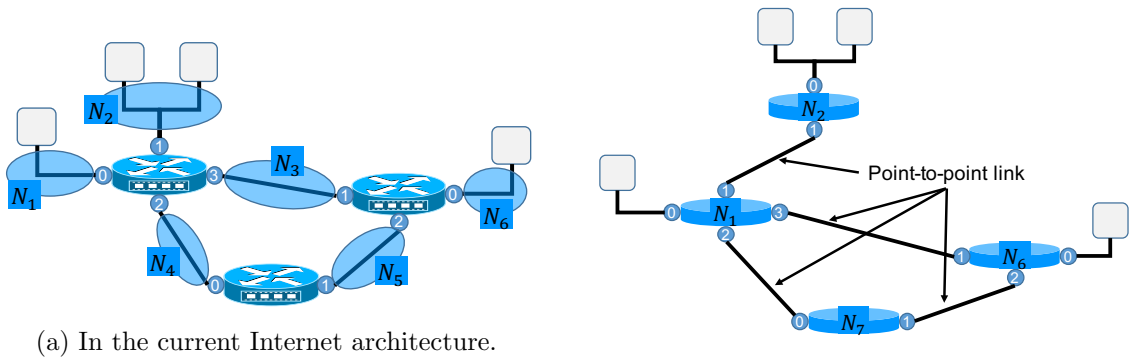
We tackle this problem by defining a confidentiality for the network addresses of the packets transferred between two communicating endpoints, where only both endpoints are the authorised entities to access the packet addresses in cleartext. We call this information security property *end-to-end network address confidentiality*, or shortly, *network address confidentiality (NAC)* classifying also network nodes as adversary. Blind Packet Forwarding (BPF) is a clean-slate security approach aiming to simultaneously establish the packet forwarding service

and to provide NAC.

This chapter presents a basic BPF construction that redesigns the packet forwarding and its associated services to blind ones which can still correctly process masked network addresses being based on a basic structure. For encryption of the network addresses, we leverage PEKS which is regarded as semantic-secure as we discussed in Section 2.1. Section 3.2 discusses BPF's security benefits and the effects of applying BPF in the current Internet architecture. Since our approach is a clean-slate design, the prototype implementation of the basic BPF design and its deployment in a real hardware testbed are presented in Section 3.3 and 3.4 in detail. Finally, we evaluate the implementation in Section 3.5.

3.1 Construction

In the current Internet architecture, a network node (router) connects multiple networks with each other by means of its interfaces. Each interface of the router has its own ID (IP address) and provides access to a network. Thus, the networks N_1, N_2, N_3, and N_4 in Figure 3.1a are interconnected with each other by means of the interfaces of a node. In contrast to that, in order to simplify the construction, a network node in our design has only one ID, and each interface of the node only has a MAC address. In this regard, the interconnected interfaces of two neighbour nodes represent the end-points of a point-to-point link (see Figure 3.1b). Thus, a node is responsible only for one local network which is represented by the ID of the node. In this way, at least two network nodes are needed in order to connect two local networks. Hence, we need two nodes in order to interconnect the local networks N_1 and N_2 with each other, as illustrated in Figure 3.1b.



(a) In the current Internet architecture.

(b) In the basic BPF design.

Figure 3.1: Network nodes.

We define a basic address structure which can be extended to more complex address structures, such as the address structure in IP, in a straightforward way. In our design, an endpoint address in cleartext is canonically represented in dotted decimal notation, which consists of two decimal numbers. The address for the endpoint X is then

$$Addr_X : N_X.H_X. \tag{3.1}$$

Here, the first decimal number N_X is the ID of the network node responsible for the local network to which the endpoint is connected, and the second decimal number H_X is the ID of the endpoint. Thus, the address fields in a packet from the source endpoint S in the local network N_S to the destination endpoint D in the local network N_D consist of $Addr_D \mid Addr_S$ (see Figure 3.2).

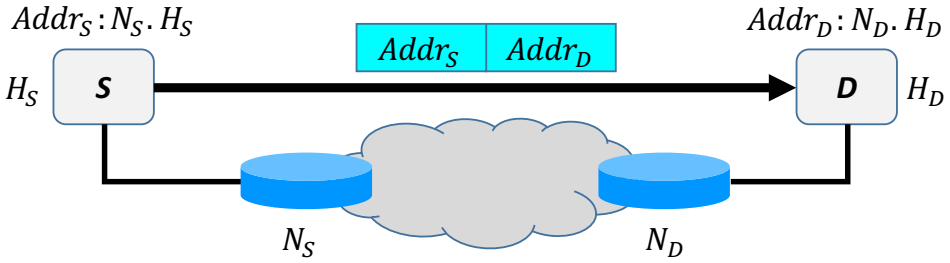


Figure 3.2: A basic address structure.

For the construction, it is assumed that all network nodes and endpoints have already generated a key pair by using PEKS and that the public keys of source and destination network nodes and endpoints have already been exchanged and bound to network nodes and endpoints, and furthermore, the communicating endpoints have already found the network addresses of each other.

3.1.1 Address masking

In our design, the cleartext address of an endpoint consists of a network and host part (see equation 3.1). For masking the address, these two parts are separately encrypted using PEKS. Thus, the masked address of the endpoint X in the local network N_X is

$$mAddr_X : E(N_X).E(H_X), \ where \tag{3.2}$$

- $E(N_X) = PEKS(N_{X_{pub}}, N_X)$ is the ciphertext for the ID of the network node, which is generated with its public key $N_{X_{pub}}$.
- $E(H_X) = PEKS(H_{X_{pub}}, H_X)$ is the ciphertext for the ID of the endpoint, which is generated with its public key $H_{X_{pub}}$.

The masked address fields in a packet from the source endpoint S in the local network N_S to the destination endpoint D in the local network N_D consist of

$$mAddr_D \mid mAddr_S \mid C(Addr_S). \tag{3.3}$$

Here, $mAddr_D$ and $mAddr_S$ are the masked addresses of the destination and source endpoint, which are generated according to equation 3.2. Moreover, $C(Addr_S)$ is the ciphertext generated by conventionally encrypting (e.g., with RSA) the source address with the corresponding public key of D (see Figure 3.3).

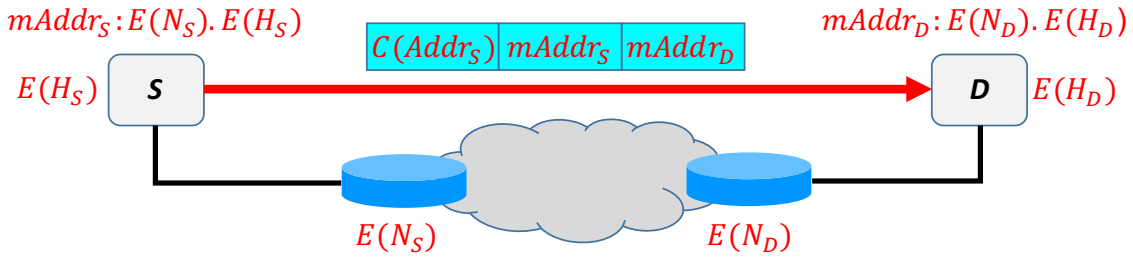


Figure 3.3: Address masking and the structure of a masked packet.

For packet generation, the source endpoint separately encrypts the network and host part of the destination address with the public key of the destination network node and destination endpoint using PEKS. In the same way, the source endpoint encrypts the parts of its own address with its own public key $H_{S_{pub}}$, and with the public key $N_{S_{pub}}$ of the node responsible for its own local network. This address field ($mAddr_S$) is needed in case that the packet cannot be forwarded to the destination. Hence, a corresponding "unreachable" message can be sent back to the source endpoint by using ($mAddr_S$) as its destination address.

By applying a public key scheme like RSA, the source endpoint address is conventionally encrypted and transferred in the third address field. This is required since the PEKS construction used for our design does not allow to decrypt ciphertexts. Thus, the destination endpoint decrypts $C(Addr_S)$ and gets the source address of the incoming packet in cleartext. To send a packet back to the source endpoint, the destination endpoint can however leverage the address field ($mAddr_S$) as the destination address of the packet. The sender and receiver endpoint can cache the encrypted addresses so that they do not have to encrypt the addresses for each packet.

3.1.2 Link layer discovery

In the basic BPF design, each network node maintains a table of links to its neighbour nodes, which is a simplified version of a LLDP neighbour table [LLD09]. At the network node N, the table entry for the link to the neighbour node N_i is

$$[LP, MAC_{LP}, MAC_{RP}], \text{ where}$$

- LP denotes N's port number through which it is connected to N_i's port with the number RP.
- MAC_{LP} is the MAC address of N's port LP.
- MAC_{RP} is the MAC address of N_i's port RP.

For the cold start, each network node broadcasts a link setup request message via all its ports (see Figure 3.4). Each of these messages contains the MAC address of the port via which the request is sent. Each neighbour network node creates a new entry for an incoming request message. This entry contains the MAC address from the message, the MAC address and the number of the local port via which the message is received. After that, each neighbour network node sends a reply message to the MAC address from the request message. The reply message contains the MAC address of the local port via which the request message is received. For each received reply message, the requesting node creates a new table entry accordingly.

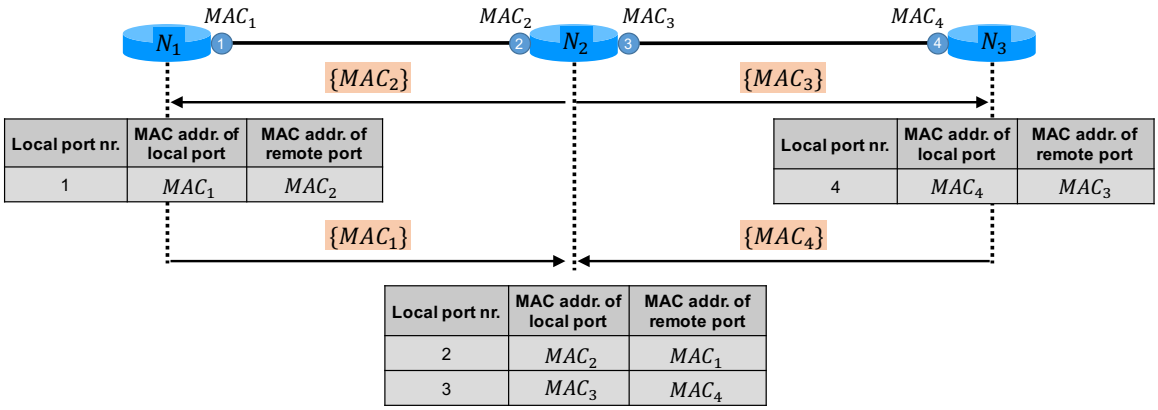


Figure 3.4: Link layer discovery.

After a link is set up, by means of keep-alive requests, the link is periodically checked whether it is still valid. If a node does not receive a keep-alive response from one of its neighbour nodes, the link to this neighbour node is declared as void.

3.1.3 Masked routing

To enable a network node to correctly forward a masked packet, its routing table has to be restructured accordingly. In our design, a *Masked Routing Table Entry (mRTE)* for the network node N is

$$mRTE_N : [(E(N), T(N)), P_N, D_N], \text{ where}$$

- $E(N) = PEKS(N_{pub}, N)$ is N's encrypted ID generated with its public key N_{pub}.
- $T(N) = Trapdoor(N_{priv}, N)$ is N's trapdoor value generated with its private key N_{priv}.
- P_N is the number of the port via which the network node N can be reached.
- D_N is the distance to the network node N.

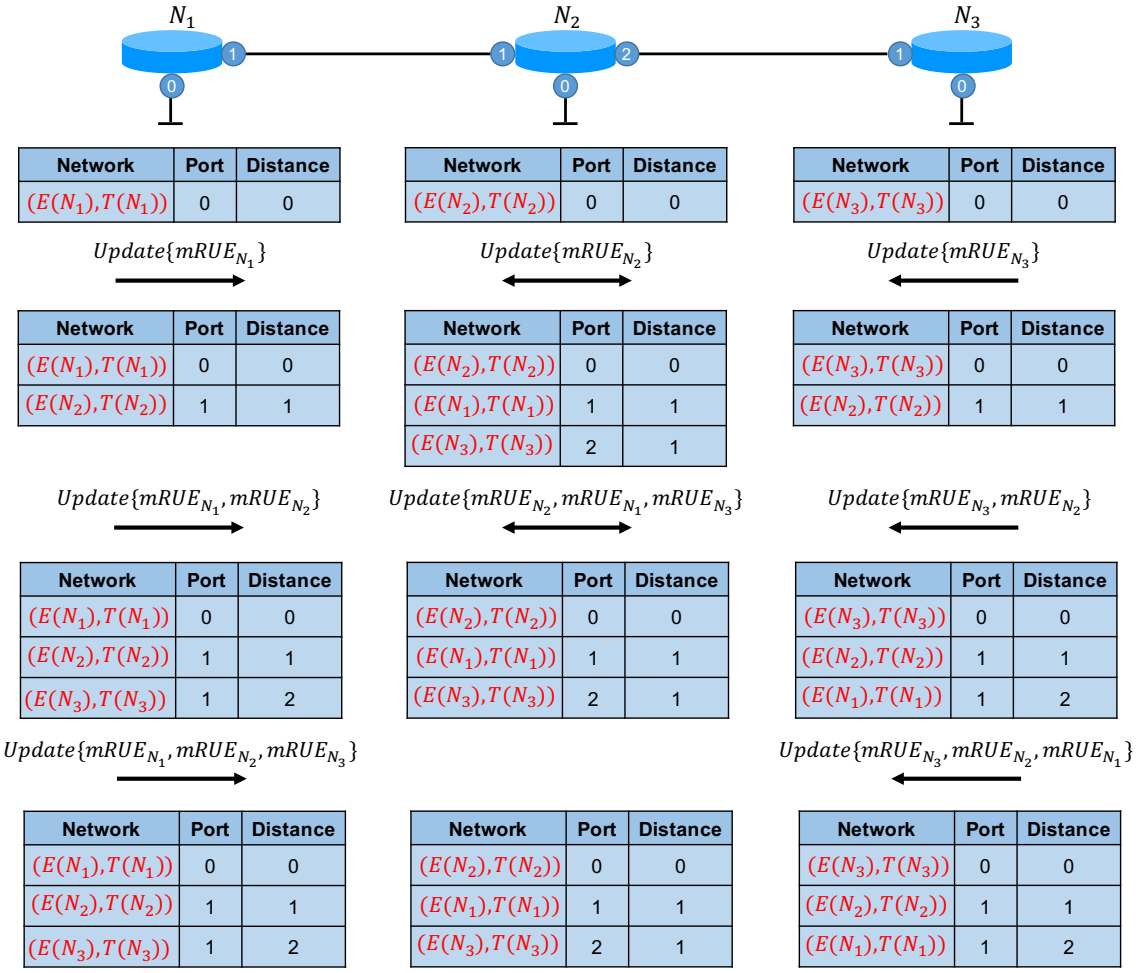


Figure 3.5: Masked routing table setup.

For a masked routing table setup, we redesign the Distance Vector Routing algorithm [BG87] to be able to work in a blind manner. For the cold start, a network node encrypts its own ID with its public key and generates the trapdoor value for its own ID with its private key by using PEKS. Moreover, the network node inserts these two values as well as the port number and the value 0 as the distance to its local network into the first routing table entry (see Figure 3.5). After that, each network node broadcasts a masked routing update message consisting of its masked routing table except the port numbers in the entries. Thus, a *Masked Routing Update Entry (mRUE)* for the masked routing table entry i, which keeps the masked routing information for the network node N_i, contains N_i's encrypted ID, trapdoor value and the distance to N_i:

$$mRUE_{N_i} : [E(N_i), T(N_i), D_{N_i}].$$

A network node maintaining a masked routing table with the size of n, broadcasts the masked routing update message

$$masked\_routing\_update\{mRUE_{N_1}, ..., mRUE_{N_n}\}$$

to all its neighbour nodes. For an incoming update message, each network node performs

$$Test(E(N_j), T(N_i))$$

for each masked routing table entry j, and for each masked routing update entry i.

- ◇ If the $Test$ method returns 1 for the entry j, i.e. the entry contains the masked routing information for the network node N_j, the network node receiving the update message compares the distances in the routing table entry and in the routing update entry with each other.
  - ◇ If $D_{N_j} > (D_{N_i} + 1)$, i.e. the new route announced in the update message is shorter[1], the routing table entry is updated with the new encrypted ID $E(N_i)$, the distance $(D_{N_i} + 1)$, and the number of the port via which the update message is received.
  - ◇ Otherwise, only the masked ID in the routing table entry is updated with the masked ID in the routing update entry in order to update the byte value of the ciphertext for node's ID.
- ◇ If the $Test$ method returns 0 for all entries, i.e. no entry exists for the node N_i so far, a new entry is created with $(E(N_i), T(N_i))$, $(D_{N_i} + 1)$, and the number of the port via which the network node has received the update message.

[1] Hop count

If the masked routing table of the network node receiving the update message is changed in this way, the network node generates a masked routing update message for its updated routing table, and it broadcast the update message to all its neighbour nodes.

If a network node detects the failure of a link to the neighbour node N_X, the node broadcasts a delete message in *Border Gateway Protocol (BGP)* [RLH06] and *Routing Information Protocol (RIP)* [Hed88]. The delete message in the masked routing contains the trapdoor value in the entry that maintains the masked routing information for the failed neighbour

$$mRTE\_delete\{T(N_X)\}$$

to all of its other neighbours, and the node removes the masked routing table entry for the neighbour node. For an incoming delete message, each network node performs

$$Test(E(N_i), T(N_X))$$

for each entry i. If the $Test$ method returns 1 for an entry, i.e. an entry exists for the node N_X, the entry is removed from the table and the delete message is broadcasted to all of the neighbours except the neighbour from which the delete message has arrived. If the $Test$ method returns 0 for all entries, i.e. no entry exists for the node N_X, the delete message is dropped.

3.1.4 Masked address resolution

When a packet arrives at a local network node, or if both communicating endpoints are connected to the same local network, the masked destination address of the packet is resolved to the MAC address of the destination endpoint and the packet is forwarded to this MAC address. In this section, we present how masked network addresses can be resolved to MAC addresses. The address resolution cache table is redesigned so that an entry for the host X with the cleartext address $Addr_X = N_X.H_X$ consists of

$$[(mAddr_X, T(H_X)), MAC_X], \text{ where}$$

- $mAddr_X = E(N_X).E(H_X)$ is X's masked address generated according to equation 3.2.
- $T(H_X) = Trapdoor(H_{X_{priv}}, H_X)$ is X's trapdoor value which is generated with its private key $H_{X_{priv}}$.
- MAC_X is the MAC address of the host X in cleartext.

In case of an empty neighbour cache, the host D aiming to resolve the masked network address $mAddr_X$ to a MAC address broadcasts a request message (see Figure 3.6). This message consists of the masked network address which has to be resolved as well as the masked network address, trapdoor value, and MAC address of the requesting host:

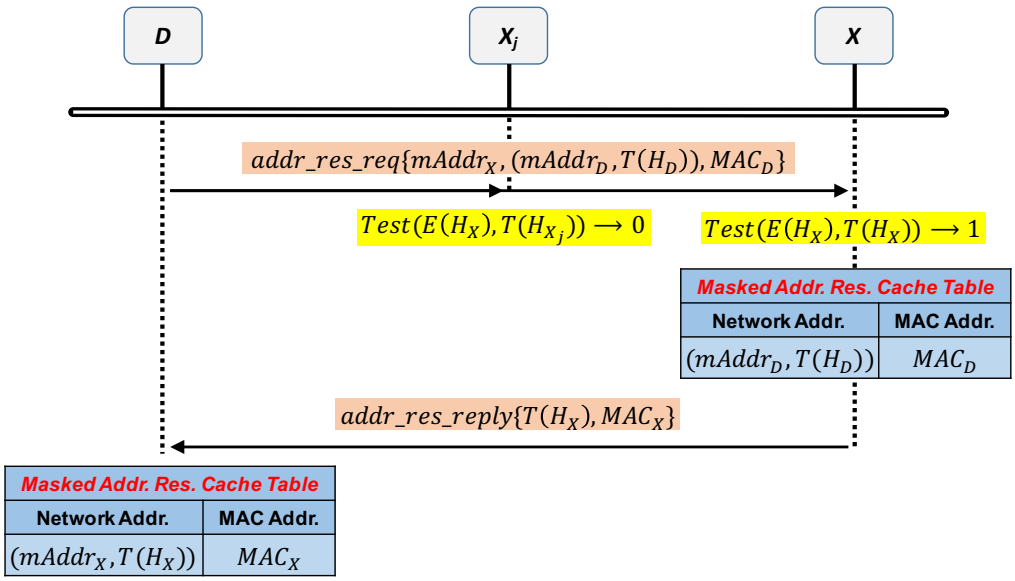


Figure 3.6: Masked address resolution.

$$addr\_res\_req\{mAddr_X, (mAddr_D, T(H_D)), MAC_D\}.$$

For the incoming request message, each host X_j in the local network performs

$$Test(E(H_X), T(H_{X_j})).$$

The $Test$ method returns 1 only at the host X, since it is the only host that can generate $T(H_X)$. While the other hosts drop the request message, the host X performs

$$Test(E(H_D), T(H_i))$$

for each cache entry i in order to determine whether it already keeps a cache entry for the host D.

- ◇ If the $Test$ method returns 1 for an entry, i.e. an entry already exists for the host D, the entry is updated with $mAddr_D$ and MAC_D.
- ◇ If the $Test$ method returns 0 for all entries, i.e. no entry exists for the host D so far, a new entry is created with the values from the request message.

Then the host X sends a reply message containing its own trapdoor value, and MAC address

$$addr\_res\_reply\{T(H_X), MAC_X\}$$

to the host D. After receiving the reply message, D creates a new entry in its table.

In case that some entries in the neighbour cache already exist, the resolution of X's masked address $mAddr_X = E(N_X).E(H_X)$ occurs by performing

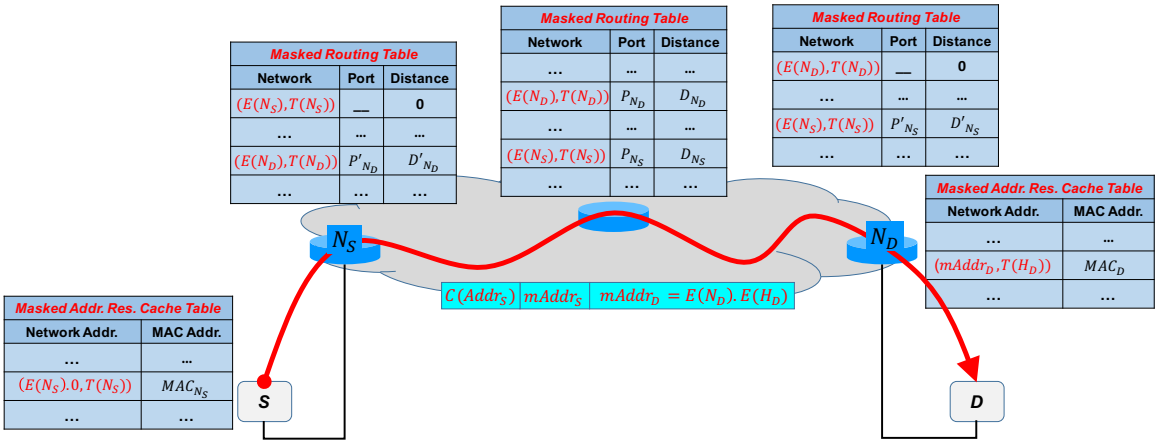


Figure 3.7: Masked packet delivery.

$$Test(E(H_X), T(H_i))$$

for each cache table entry i. The masked address is then resolved to the MAC address at the entry for which the $Test$ method returns 1. Performing a binary comparison of a masked address with the masked addresses in the cache table before executing the $Test$ method can speed up the masked address resolution. If no entry can be found in this way, the masked address is resolved as described above.

3.1.5 Masked packet delivery

The endpoint S in the local network N_S wants to send a packet to the endpoint D in the local network N_D (see Figure 3.7). We assume that all masked address resolution and routing tables are already set up. The source endpoint creates the packet with the masked addresses according to equation 3.2. Thus, the source endpoint gets the packet

$$< mAddr_D \mid mAddr_S \mid C(Addr_S) >, \text{ and } mAddr_D = E(N_D).E(H_D).$$

- ◇ If $N_S = N_D$, i.e. the source and destination endpoint are connected to the same local network, the masked destination address is resolved to the MAC address of the destination endpoint and the packet is sent to this MAC address (see Section 3.1.4).
- ◇ Otherwise, the source endpoint resolves the masked ID of the network node N_S to its MAC address and sends the packet to the network node.

For the incoming masked packet, a network node performs

$$Test(E(N_D), T(N_i))$$

for each masked routing table entry i.

- ◇ If the $Test$ method returns 1 for an entry, the source MAC address of the packet is updated with the MAC address of the port mapped in that routing table entry, while the destination MAC address of the packet is replaced with the MAC address mapped in the neighbour table entry for that port. After that, the packet is forwarded via this port.

- ◇ If the $Test$ method returns 0 for all entries, it means that no masked routing table entry for the destination network node exists. In this case, the packet is dropped, and a corresponding "unreachable" message can be sent back to the source endpoint by using the second address field ($mAddr_S$) as its destination address.

Eventually, the packet arrives at the network node responsible for the local network N_D. It also performs the $Test$ method, and detects that the packet is addressed to its own local network. Thus, the node resolves the masked destination address to the MAC address of the destination endpoint D and sends the packet to it.

3.2 Analysis

Information confidentiality is one of the most important security properties in communication. The goal here is to disclose information only to authorised entities. Ideally, only communicating endpoints have access to information transferred between them. However, a in-network service could also require access to that information in order to accomplish its task. Thus, either the in-network service has inevitably to be classified as an authorised process having access to the information in cleartext, or the communicating endpoints cannot use the in-network service anymore. Network packet addresses are the basic information transferred between two communicating endpoints, and the packet forwarding service is the fundamental in-network service needed by the endpoints.

Beside third parties, e.g., potential attackers sniffing network packets, network address confidentiality (NAC) also classifies network nodes forwarding packets between two communicating endpoints as adversary. Thus, only the source and destination endpoints are the authorised entities having access to the network packet addresses in cleartext. Here, we want to point out that communicating endpoints do not consider each other as adversary. NAC includes end-to-end transmission confidentiality and *end-to-end processing confidentiality* for the network addresses of packets transferred between two communicating endpoints. While end-to-end transmission confidentiality masks network packet addresses from third parties during transmission, end-to-end processing confidentiality masks the addresses from network nodes processing packets transferred between two communicating endpoints.

Confidentiality of information, which can be used to identify a subject, can deduce a certain degree of subject anonymity against adversaries. In Section 2.4.1, we have discussed the anonymity properties defined by [PH05]. In the context of this thesis, subjects are communicating endpoints, the items of interest are network packets transferred between them, identities are their network addresses in the packets, and adversaries are third parties and network nodes taking part in the packet transfer. Thus, NAC deduces the following anonymity properties:

- **Sender/recipient unlinkability:** Network packets (traffic) captured by an adversary cannot be linked to their source/destination endpoint on the basis of their network addresses.
- **Relationship unlinkability:** An adversary cannot link network packets (traffic) to the communicating endpoint pairs on the basis of their network addresses so that it is masked who communicates with whom. This can be also called traffic flow confidentiality masking the source-destination traffic pattern.

As stated in [PH05], sender or recipient unlinkability implies relationship unlinkability. This means that an anonymity system providing sender or recipient unlinkability also supplies relationship unlinkability.

We want to emphasize that NAC aims to mask only network packet addresses from adversaries. Thus, NAC does not consider other traffic patterns such as packet quantities, sizes and interarrival times. By means of these traffic patterns, an adversary can perform traffic analysis attacks [DC07] in order to correlate network packets with each other. In this way, the adversary can detect a traffic flow. However, the adversary still needs network packet addresses in cleartext in order to assign the traffic flow to its source and destination endpoints. Thus, network packet addresses are still most crucial traffic patterns.

It is also to be pointed out that we do not consider end-to-end user information confidentiality. In order to achieve this property too, user payloads have to be conventionally encrypted end-to-end as is the case in TLS.

3.2.1 Network address confidentiality in related works

Before analysing NAC and the deduced anonymity properties in BPF, we first discuss NAC in related approaches with regard to our security model defining all entities except the source and destination endpoints of network packets as adversary. To the best of our knowledge, IP encapsulating security payload protocol in tunnel mode [Ken05] is the only approach aiming to achieve NAC primarily.

Since NAC deduces the anonymity properties discussed above, we also discuss approaches aiming to achieve these properties. Anonymity systems can be classified in high- and low-latency anonymity systems as is the case in [EY09]. High-latency anonymity systems (e.g., [Cha81], [SDS02], and [Dan03]) are designed to be applied for non-interactive applications tolerating delays of several hours or more, such as email. Since BPF aims to support transparency to applications, high-latency anonymity systems fall out of scope of our discussion. In [RW10], Ren *et al.* divide low-latency anonymity systems into the following categories: Mixnet-based schemes, routing-based techniques, peer-to-peer networks. Thus, we discuss the anonymity properties of pioneer approaches from each category. These approaches are already sketched in Section 2.4 in more detail.

3.2.1.1 IP Encapsulating Security Payload (ESP) protocol in tunnel mode

ESP protocol in tunnel mode [Ken05], shortly ESP, encrypts an IP packet including its IP header, and puts the encrypted packet in a new IP packet with a new IP header containing the IP addresses of IPsec peers (security gateways) connecting the private networks with the public network. Thus, the original packet with its ultimate source and destination addresses are transferred between the security gateways in encrypted form. ESP classifies the private networks including the security gateways as trusted and the public network as untrusted. Hence, security gateways and network nodes within the private networks are authorised entities having access to the ultimate source and destination addresses of the original packet in cleartext. In contrast to that, BPF classifies no entity except the source and destination endpoints as authorised thus trusted.

ESP provides NAC for the ultimate source and destination addresses of the original packet within the public network, but not within the private networks, and security gateways still have access to the addresses in cleartext. Thus, ESP provides a restricted form of NAC for the ultimate source and destination addresses. Moreover, the addresses of the security gateways in the outer header are transferred in cleartext so that NAC is not supplied for these addresses.

Although ESP does not aim to provide any anonymity property explicitly, the restricted NAC in ESP deduces sender/recipient and relationship unlinkability within the public network with regard to the ultimate source and destination addresses of packets. However, adversaries within the private networks can link packets to their ultimate source and destination endpoints and knows who communicates with whom on the basis of their addresses. In addition, the anonymity properties do not apply to security gateways. Thus, any adversary

within the public network can link traffic to the security gateways and detect which security gateways communicate with each other.

By means of packet padding, ESP protects network packets transferred between security gateways also against traffic analysis attacks being based on correlation of packets with regard to their sizes. In [KTB+07], Kiraly *et al.* have presented an ESP extension masking further traffic patterns such as packet quantities and interarrival times.

3.2.1.2 Tor

Tor [DMS04] is a mixnet-based overlay network aiming to anonymise TCP connections. Onion routers (ORs) in the network maintain TLS connections to each other. Tor users run onion proxies (OPs) communicating with ORs using TLS connections. The OP of an user aiming to communicate anonymously selects an ordered set of usually three ORs ($OR1$, $OR2$, $OR3$). After that, the OP negotiates a symmetric key with $OR1$ (entry OR), and thus builds a circuit. Each newly created circuit is used to iteratively extend the circuit to the next OR until reaching the last OR (exit OR). Thus, the OP shares a symmetric key with each OR ($OR1_{SK}$, $OR2_{SK}$, $OR3_{SK}$), and each OR on the circuit learns its predecessor OR/OP (IP address) and successor OR (IP address), which are mapped to circuit IDs that are randomly selected hop-wise.

To send a TCP payload to given destination address and port, the OP packs them as the payload of a Tor frame (cell) and encrypts the payload layer-wise with each shared key. Thus, the OP gets the onion $E_{OR1_{SK}}(E_{OR2_{SK}}(E_{OR3_{SK}}(payload)))$. The header of the cell contains the ID of the circuit to be used. The cell is then sent through the circuit. Each OR receiving the cell decrypts the outer layer of the onion with its symmetric key mapped to the circuit ID of the incoming cell, updates the circuit ID for the next hop and sends the cell containing the remaining onion as payload to the successor OR mapped to the new circuit ID. Thus, $OR3$ gets the cell payload containing the TCP payload that is sent to the specified destination address. Thus, the destination endpoint considers $OR3$ as the source endpoint of the payload.

To reply to the actual source endpoint, the destination endpoint sends the reply TCP payload to $OR3$ that packs the TCP payload and its source address and port as cell payload and encrypts the cell payload with its key. After that, $OR3$ sends the cell to its predecessor OR. Each OR on the circuit puts its own encryption layer to the onion and the OP gets the onion $E_{OR1_{SK}}(E_{OR2_{SK}}(E_{OR3_{SK}}(payload)))$. Finally, the OP decrypts the onion layers with the shared keys to get the TCP payload.

Network packets carrying Tor cells from/to the source endpoint have its address in cleartext. Thus, network nodes forwarding the network packets between the source endpoint and the entry OR and the entry OR itself are authorised entities having access to the network address of the source endpoint in cleartext. The same with regard to the network address of the destination endpoint applies to the exit OR and network nodes transferring network packets between the destination endpoint and the exit OR. However, no single network node and OR alone have access to the cleartext network addresses of the source and destination endpoints simultaneously. Nonetheless, this is possible for a global adversary having the ability to sniff the incoming/outgoing network packets to/from the entry/exit OR. But the adversary has to correlate these network packets with each other, which is also possible by means of traffic analysis techniques stated in [DC07]. Hence, NAC is not provided by Tor.

Since adversaries have access to the cleartext network addresses of the source and destination endpoints, they can link network packets (traffic) to the source/destination endpoint on the basis of their network addresses. Thus, sender/recipient unlinkability is not provided by Tor. In contrast to that, Tor aims to supply relationship unlinkability anyway. Moreover, Tor also prevents traffic analysis attacks being based on packet size by means of cells of a fixed size. However, a global adversary can link traffics to communicating endpoint pairs by means of timing analysis attacks [LRWW04].

Since the destination endpoint considers the exit OR as the source endpoint of the traffic, the actual sender remains anonymous for the recipient. In addition, Tor supplies hidden services masking the network addresses of servers accessible to clients requesting their services through the Tor network. However, Øverlier *et al.* have presented an attack which can be leveraged to disclose the true host of a hidden service [OS06].

3.2.1.3 Crowds

The Crowds system [RR98] is a routing-based technique for anonymous web browsing. A crowd can be considered as a collection of endpoints running special web proxies called jondos. When a jondo is started, the jondo registers itself at a server called blender. The new jondo gets the membership of the crowd and is reported to other jondos. Moreover, the new jondo shares an encryption key with each other jondo, and the connection between each two jondos is encrypted.

To send a request to a given web server, the jondo randomly picks a jondo in the crowd, caches the selected jondo as its successor, and forwards the request to the selected jondo. Here, it is also possible that the initiator jondo selects itself. When receiving a request, a

jondo either forwards the request to another jondo selected at random, or it submits the request to the web server. The probability for forwarding the request to a further jondo is more than fifty percent. If the jondo decides to forward the request, it caches the jondo from which the request came as its predecessor, and the jondo to which the request will be forwarded as its successor. Subsequent requests from the initiator jondo to the web server take the same path, and server replies travel back the path in reverse.

Each jondo on the path, third parties sniffing the traffic between the last jondo and the web server, and network nodes transferring the packets between the last jondo and the web server have access to the web server's address in cleartext. Although third parties and network nodes between the initiator jondo and the second jondo and the second jondo itself have access to the cleartext address of the initiator jondo, they do not know with certainty whether the request comes from the initiator jondo or from a jondo that just forwards the request. However, the actual initiator jondo of a request can be disclosed by means of predecessor attacks presented in [WALS04]. In summary, it can be stated that the Crowds system does not provide NAC.

Since each jondo on the path knows which web server is requested, recipient unlinkability is not the case in the Crowds system. Because of the vulnerability to predecessor attacks, web requests can be linked to actual initiator jondos so that sender unlinkability is also not provided. Due to the ability to link requests to clients and servers, their relationships are thus linkable too.

3.2.1.4 Tarzan

Tarzan [FM02] is a peer-to-peer network aiming to anonymise IP packets. In the Tarzan network, peers relay IP data packets for each other. To discover the network peers and to get their public keys, a new peer selects a peer from a set of bootstrap peers at random and learns the neighbour peers of that peer. The new peer then selects another peer from the new set of known peers at random and repeats the process analogously. After discovering the network, the new peer selects a certain number of peers (mimics) with which the peer exchanges dummy traffic (mimic traffic).

To build up an anonymous tunnel for data packet transfer, the initiator peer randomly selects an ordered series of peers $\{p_1, ..., p_n\}$ from its mimic set. The initiator peer then sends an establishment request to each peer p_i. The request contains the forward and backward symmetric keys (fk_{p_i}, bk_{p_i}), the IP addresses of the predecessor and successor peers (p_{i-1}, p_{i+1}), and the pairwise flow identifiers to tag data packets to the forward and backward flows. More-

over, the request is encrypted with the public key of the peer p_i and relayed as a normal data packet from p_1 through p_{i-1}.

Once the tunnel is established, the initiator peer can begin with the transfer of IP data packets from an application. To send a packet, the initiator peer first rewrites the real source address of the packet with a random address and creates the onion $E_{fk_{p_1}}(...E_{fk_{p_n}}(packet)...)$ tagged to the forward flow. After that, the peer encapsulates the onion and its tag in a UDP packet and sends it to p_1. Upon receiving the UDP packet, each peer p_i decrypts the outer layer of the onion with its symmetric key fk_{p_i}, retags the remaining onion, encapsulates the result in a new UDP packet, and sends the packet to the successor peer. Eventually, the packet arrives at the last peer that decrypts the last layer of the onion and gets the IP packet of the initiator peer. Finally, the last peer on the circuit writes its IP address into the source address field of initiator's IP packet and sends it to the destination endpoint. The reply packets take the same tunnel path in reverse, and each peer puts its encryption layer using its backward symmetric key. Thus, the initiator peer gets the onion $E_{bk_{p_1}}(...E_{bk_{p_n}}(packet)...)$.

The last peer on the path, eavesdroppers between the last peer and the destination endpoint, and network nodes transferring the packets between them have access to the cleartext address of the destination endpoint. The same applies to the traffic between the initiator peer and the first peer on the circuit. However, adversaries cannot state with certainty from which peer the packets originate. Moreover, the real data traffic is masked by means of mimic traffic. Nonetheless, an adversary controlling the first and last peer on the path can disclose the actual initiator peer by means of the intersection attack [WALS02]. Thus, the adversary has access to the cleartext addresses of the destination endpoint and the initiator peer simultaneously. Hence, Tarzan does not supply NAC.

Since the last peer on the circuit knows which endpoint is the destination, it can readily link packets to their destinations. Moreover, the intersection attack makes it possible to link packets to their senders. Thus, sender/recipient unlinkability is not provided by Tarzan. Furthermore, an adversary, which controls the first and last peer on the path and performs the intersection attack, is able to link packets to their senders and recipients. Hence, relationship unlinkability is also not supplied by Tarzan.

3.2.2 Network address confidentiality in BPF

BPF is a clean-slate approach aiming to provide NAC. In Section 3.1, we have presented a basic BPF construction that redesigns the packet forwarding service and related services to blind ones which can still correctly process masked network addresses being based on a

basic structure. For encryption of the network addresses, we have leveraged PEKS which is regarded as semantic-secure as discussed in Section 2.1. Moreover, the encryption function of PEKS is not deterministic. This means that the function outputs different ciphertexts (masked network addresses) for each encryption of the same cleartext (network address) with the same public key.

In contrast to the approaches discussed above, network packet addresses in our design are transferred as well as processed end-to-end in encrypted form. Although the addresses of the packets are encrypted end-to-end, the packets can be still forwarded in the right direction so that network nodes do not have to become entities authorised to access the packet addresses in cleartext, anymore. Thus, no entity except the source and destination endpoints, i.e. no adversary has access to the cleartext network addresses of the packets transferred between both endpoints. Hence, BPF makes it possible to simultaneously establish the packet forwarding service and to provide NAC. Moreover, network node IDs in routing update messages are also handled in encrypted form. In this regard, NAC in BPF also applies to network nodes and their IDs.

In the low-latency anonymity systems discussed above, system participants relay packets through a pre-built path to hide the identities (network addresses) of actual source endpoints. In [WALS02], Wright *et al.* have presented a general model of attacks to which these systems are vulnerable. This model is based on the fact that each node on the path knows the addresses of its predecessor and successor. Moreover, the entry and exit node of the path have access to the addresses of the actual source and destination endpoints. By means of traffic analysis being based on traffic patterns such as packet interarrival times, a global adversary controlling the entry and exit node can correlate the incoming and outgoing packets with each other. Thus, the adversary can detect a traffic flow and assign it to the addresses of the source and destination endpoints. In this way, the adversary can disclose who communicates with whom.

In contrast to that model, network nodes in BPF are not aware of the identities of each other. Thus, network nodes do not know the previous and next hop of a packet. Moreover, network nodes, even the source and destination network nodes, do not have access to packet addresses in cleartext. Thus, even if a traffic flow is detected, it cannot be linked to its source and destination endpoints on the basis of their network addresses. Furthermore, if packet addresses are re-encrypted at random, packets belonging to the same communicating endpoint pair cannot be linked to the same traffic flow with certainty. Since BPF does not aim to mask other traffic patterns such as packet quantities, sizes, and interarrival times, the packets can

be correlated with each other using these patterns. However, the packets still differ in the source-destination pattern at random. Thus, traffic flows cannot be distinguished from each other with certainty.

A local adversary, i.e. a single network node or eavesdropper on that network node sees packets containing encrypted addresses. Thus, the adversary does not know from which endpoint a packet originates and to which endpoint a packet is addressed. Hence, BPF provides sender unlinkability and recipient unlinkability against a local adversary. Since one of these properties implies relationship unlinkability, the adversary also cannot link a packet to a communicating endpoint pair. Moreover, confidentiality of traffic flows is achieved, if packet addresses are re-encrypted at random. These properties do not only apply to endpoints, but also to local networks. This means that a packet cannot be linked to a local network as its source or destination, and the relationship of communicating local networks is not linkable. However, if a packet originates from an endpoint connected to the adversary network node, the adversary knows the source local network of the packet. But the adversary still does not know the source and destination endpoints of the packet and its destination local network. The same with regard to the source local network applies to the packet, if the adversary is the destination network node.

Sender/recipient and relationship unlinkability with regard to local networks do not apply against a global adversary controlling the source and destination network node of a packet. Thus, the adversary can link the packet to its source and destination local networks. In this way, the adversary can restrict the set of endpoints to which the packet can possibly be linked as its source and destination endpoints. But the packet is still not exactly linkable to its actual source and destination endpoints on the basis of their network addresses. Thus, BPF with regard to sender/recipient and relationship unlinkability is resistant to a global adversary. Because of a restricted set of possible communicating endpoints, traffic analysis is much easier for the adversary. Therefore, it is recommended to re-encrypt addresses for each packet so that traffic flows between endpoints connected to the adversary network nodes still remain undistinguishable from each other.

The byte values of the masked addresses of a packet do not change during the entire transmission of the packet. Thus, a stronger global adversary, which controls all network nodes participating in the transfer of a packet between two endpoints, can disclose the entire route of the packet by means of a costly process. Here, the adversary monitors all ports of all nodes and compares the masked addresses of all incoming and outgoing packets byte by byte with each other. As the weak global adversary above, the strong global adversary knows from

which local network a packet originates and to which local network a packet is addressed. Thus, the strong adversary knows which local networks communicate with each other. But the packet is still not linkable to its actual source/destination endpoint and to a communicating endpoint pair. Hence, sender/recipient and relationship unlinkability is provided even against the strong global adversary. As discussed above, to make traffic flows undistinguishable from each other, addresses are to be re-encrypted for each packet.

BPF makes it also possible that the network address of a source endpoint can remain anonymous for the destination endpoint just by omitting the third address field ($C(Addr_S)$) in equation 3.3. Thus, the destination endpoint receives a packet only with the masked address fields $mAddr_D \mid mAddr_S$. Since the PEKS construction used for our design does not allow to decrypt ciphertexts, i.e. the encryption is not invertible, the destination endpoint cannot decrypt the masked source network address ($mAddr_S$) in order to get it in cleartext. In this case, to send a packet back to the source endpoint, the destination endpoint can set the value in the second address field ($mAddr_S$) as its destination address.

For a masked communication, a source endpoint needs a tuple consisting of public keys of the destination endpoint and destination network node. Therefore, like other systems, which are based on an asymmetric key encryption, our design requires an infrastructure to exchange and certify public keys. For this, a Public Key Infrastructure (PKI) in each network domain can be utilised so that each domain PKI manages the certificates of the network nodes and endpoints in its own domain. In case of masked cross-domain communication, the root Certification Authorities (CAs) of each domain can certify each other by using pairs of CA cross-certificates.

As discussed above, BPF masks only the source/destination traffic pattern. Although packets cannot thus be correlated with each other with certainty, other patterns of packets, such as their sizes and interarrival times can possibly reveal information about applications running at the communicating endpoints. To hide these patterns too, the basic BPF design can be expanded by the packet padding/timing and traffic padding techniques [Tim96] in a straightforward way.

BPF does not aim to provide integrity of network packet addresses. Hence, any adversary can unnoticeably manipulate masked packet addresses. E.g., an adversary can secretly redirect packets to itself by replacing the masked packet addresses with its own masked address. Moreover, BPF advances such an attack, since masked source address of a packet is not decryptable. Thus, an endpoint receiving a masked packet cannot verify whether the packet

actually originates from the endpoint whose address is conventionally encrypted and given in the third address field of the packet (see equation 3.3). However, this can be resolved by expanding the masked packet header by a integrity check value as in ESP. Nonetheless, the adversary cannot target a specific endpoint or a certain communicating endpoint pair, since packets cannot be linked to their source/destination endpoint and thus to a communicating endpoint pair. The same with regard to masked routing information applies to routing update messages. Thus, we can state that BPF, due to its traffic flow confidentiality, hides potential targets against active attacks such as address spoofing, man-in-the-middle and repetition attack.

3.2.3 Applying BPF in the current Internet architecture

In the basic BPF design, a masked endpoint address consists of two parts. The first part is the ciphertext of the network node ID, which has been encrypted with the public key of the network node, and the second part is the ciphertext of the endpoint ID, which has been encrypted with the public key of the endpoint.

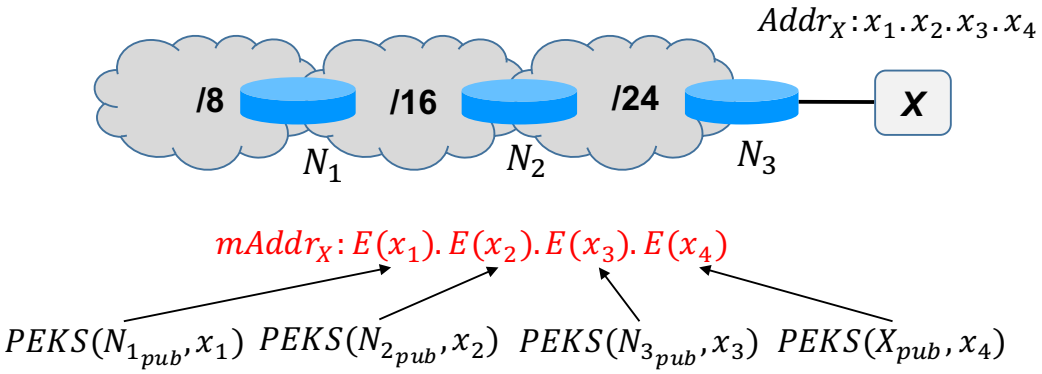


Figure 3.8: Masking of an IPv4 address according to the basic BPF design.

When this address structure is extended to a more complex one like IPv4 with conventional address classes, the address parts of two communicating endpoints have to be encrypted byte by byte according to the netmasks (i.e., without subnetting) used on the route between them (see Figure 3.8). For that, we have to establish a signalling mechanism which can be used by the communicating endpoints to get the required netmasks. Moreover, such a proceeding for address masking leads to bandwidth problems, since only one address field in a masked packet could require an enormous amount of packet sizes in this way (see Table 3.3). Furthermore, the current flat structure of IPv4 addresses boosts this issue even more.

In the basic BPF design, a source endpoint requires the public key of gateway nodes (i.e., nodes at the border of two neighbouring domains) on the route to the destination endpoint. Thus, we have to design an infrastructure which exchanges and certifies the public keys. Since the source endpoint requires all public keys of the gateway nodes, the flexible and dynamic management of such an infrastructure would be very costly.

Since routing table entries are encrypted, they cannot be aggregated. Therefore, conventional super- and subnetting is not supported. This would lead to critical scalability problems for maintaining routing tables. Thus, while demonstrating the general feasibility to provide NAC and its deduced security properties, the basic BPF design is not suitable to be deployed in the current Internet architecture.

3.3 Implementation

In the prototype implementation of the basic BPF design, a cleartext endpoint address consists of two bytes, where the first octet identifies the ID of the network node responsible for the local network and the second octet identifies the ID of the endpoint in that local network.

We have implemented a new basic network packet header structure consisting of the masked destination and source address fields according to equation 3.3, while the third address field is omitted in our implementation. To encrypt network addresses and to generate key pairs and trapdoor values we have leveraged the PEKS library [ALPK07].

A masked network packet encapsulating user data in its payload gets a new Ethernet payload type in our implementation. To realise the functionalities designed for BPF we have defined multiple new Ethernet payload types for the network control messages: Link layer discovery packet, masked routing update packet, and masked address resolution packet.

3.3.1 Network side

For the implementation on the network side, we have utilised the SDN protocol OpenFlow 1.3 by expanding the OpenFlow controller NOX by the component *Basic_BPF* which manages network nodes called datapaths in the context of OpenFlow. We have sketched OpenFlow and NOX in Section 2.3. *Basic_BPF* is a subclass of the class *Component* from NOX. By means of the functions *handle_datapath_join_event()* and *handle_packet_in_event()*, *Basic_BPF* is a listener of the *OpenFlow-Datapath-Join* and *OpenFlow-Packet-In* events which are resolved by registering a new datapath, and by receiving a packet at a datapath.

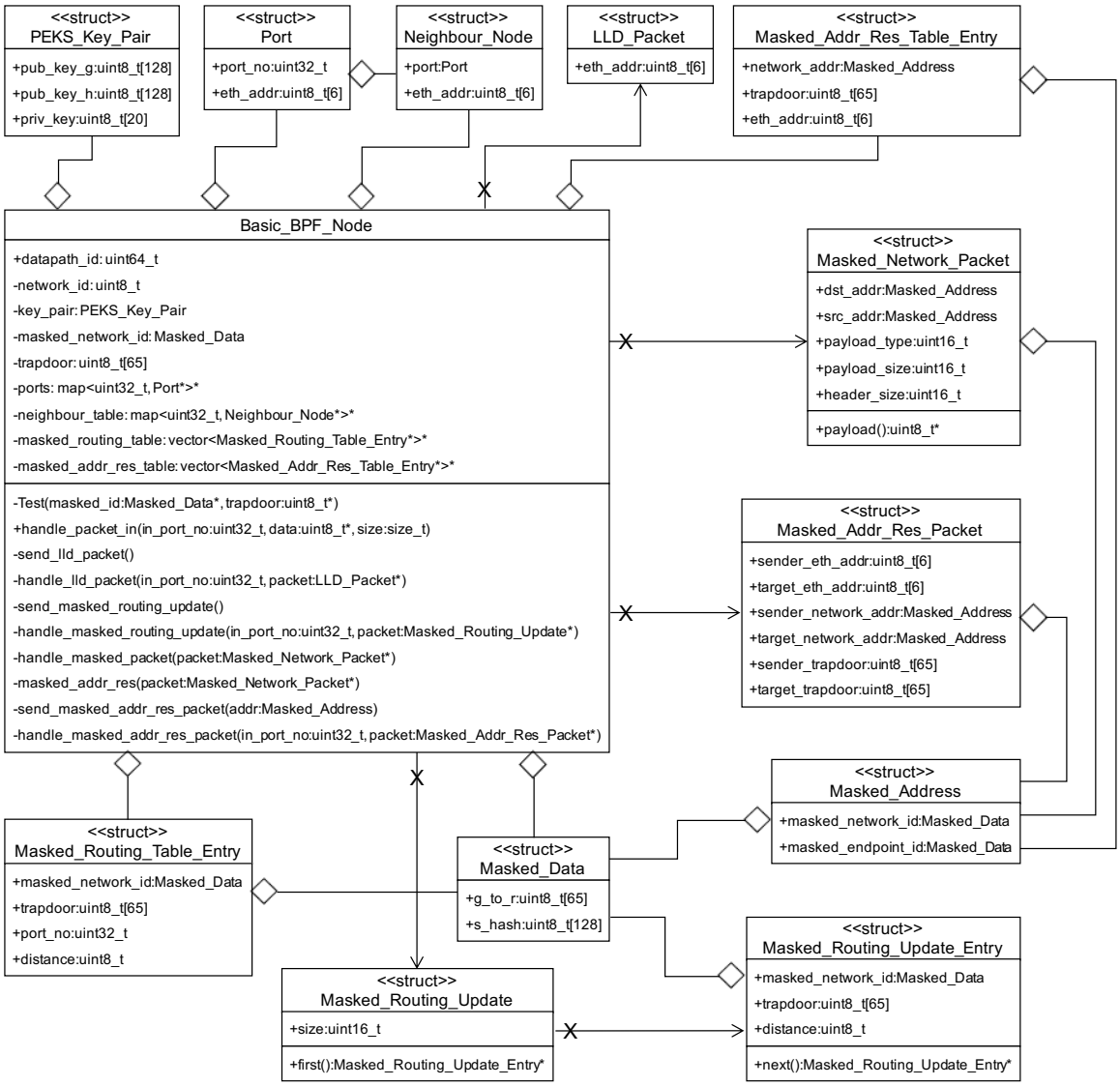


Figure 3.9: UML diagram of *Basic_BPF_Node* and the associated data structs.

The C++ class *Basic_BPF_Node* implements several attributes to keep the OpenFlow-specific ID, the network ID in cleartext, the key pair, the encrypted ID, trapdoor value, the ports, the neighbour table, the masked routing table, and the masked address resolution cache table of a blind network node. Moreover, this class implements multiple functions to realise the functionalities of a blind network node.

An UML diagram of the class *Basic_BPF_Node* and the associated data structures is given in Figure 3.9. Thus, the controller component *Basic_BPF* maintains a list of objects instantiated from this class. Each class object is identified with an OpenFlow-specific ID of the associated datapath (see Figure 3.10).

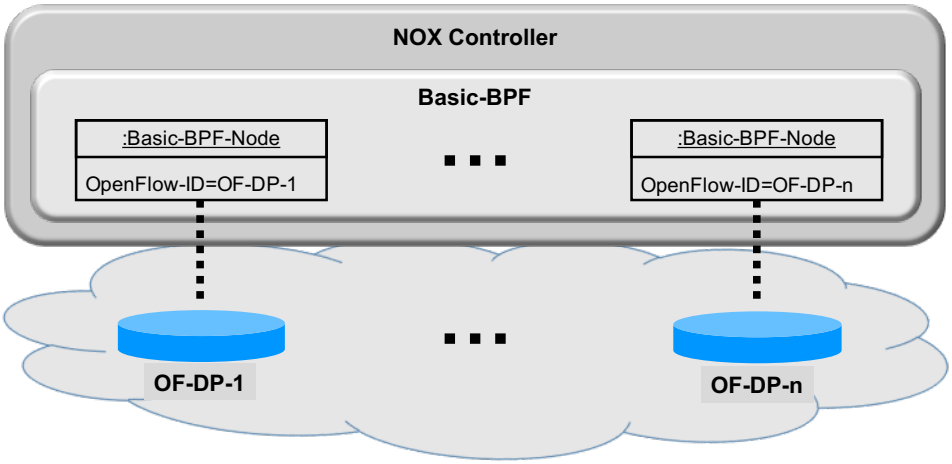


Figure 3.10: Managing of blind network nodes.

3.3.1.1 Transactions between datapaths and the controller

If a new network node is added, it registers itself with its OpenFlow-specific ID at the controller, and the controller component *Basic_BPF* instantiates a new object from *Basic_BPF_Node*, which represents the new network node at the controller. For the new network node, the controller component instructs the datapath to define flows which take care of sending packets with the Ethernet types defined for our implementation to the controller.

For an incoming packet matching a flow, a datapath sends an OpenFlow-Packet-In message to the controller. This message contains the OpenFlow-specific ID of the datapath and the packet. After receiving the message, the controller component determines the object representing the network node by means of the OpenFlow-specific ID contained in the message. After that, the function *handle_packet_in()* is called on the object.

To send a packet from a datapath, the controller component sends an OpenFlow-Packet-Out message to the datapath. This message contains the packet and the number of the port via which the packet has to be sent. Figure 3.11 visualises the transaction between a network node and the controller for sending and handling a masked packet. Thus, each incoming packet has to be sent to the controller in order to be handled, since the flow match field types currently defined in the OpenFlow specification rely on the IP packet structure.

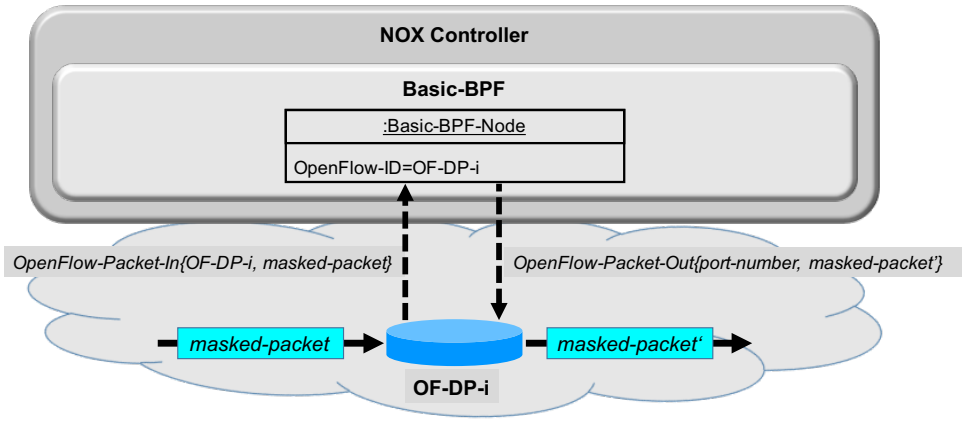


Figure 3.11: Transaction between a network node and the controller.

3.3.1.2 Blind network setup

After generating a key pair and trapdoor value, and masking the identity of a new network node in the constructor of *Basic_BPF_Node*, a link layer discovery packet is generated and broadcasted by calling the function *send_lld_packet()* on the object representing the new network node. A network node receiving a link layer discovery packet sends it to the controller for handling the packet, which is implemented in the function *handle_lld_packet()*.

After setting up the neighbour table of the new node, the controller component *Basic_BPF* calls the function *send_masked_routing_update()* for generating and broadcasting a masked routing update in order to set up the masked routing table for the new network node, and to update the masked routing tables of the other network nodes with the masked routing information for the new network node. The handling of an update packet received by a network node is realised in the function *handle_masked_routing_update()*. Thus, the functions for sending and handling link layer discovery and masked routing update packets implement the functionalities to set up a blind network.

3.3.1.3 Handling of an incoming masked network packet

For an incoming masked network packet, a network node sends an OpenFlow-Packet-In message to the controller, which contains the packet and the OpenFlow-specific ID of the sender datapath. By means of this ID, the controller component first finds out the object representing the network node, and calls the function *handle_masked_packet()* on the object. The flowchart of this function is given in Figure 3.12. The function performs *Test()* which takes the network part of the masked destination address and the trapdoor value in each masked routing table entry as parameters.

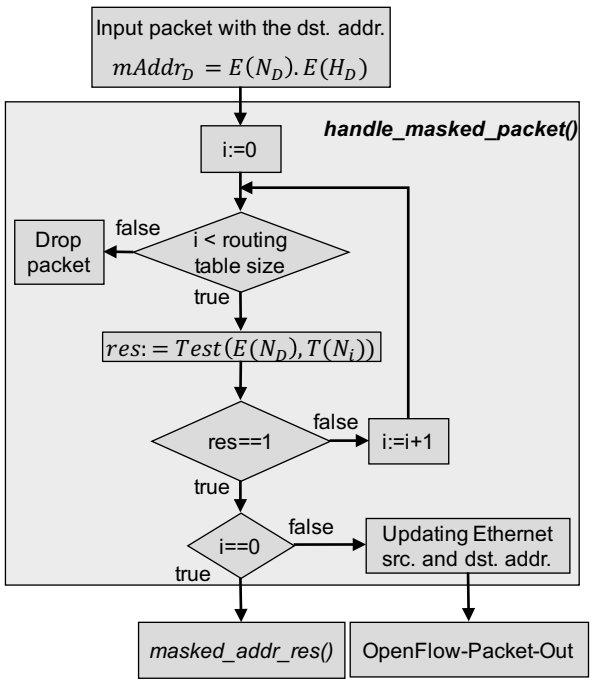


Figure 3.12: Flowchart for handling masked network packet.

- ◇ If *Test()* returns 1 for a routing table entry $i > 0$, the Ethernet source and destination addresses of the packet are updated, and an OpenFlow-Packet-Out message is sent to the network node which sent the OpenFlow-Packet-In message. The OpenFlow-Packet-Out message contains the updated packet and the number of the port via which the packet has to be forwarded. After receiving the message the network node forwards the packet via the port ordered by the controller.
- ◇ If $Test()$ returns 0 for all entries, the packet is dropped.
- ◇ In case that the packet is addressed to the local network for which the node is responsible, i.e. $Test()$ returns 1 for the first routing table entry, the function *masked_addr_res()* is called.

The *masked_addr_res()* function, whose flowchart is given in Figure 3.13, first performs a binary comparison of the masked destination address with the masked addresses in the address resolution table. If no entry can be found in this way, the *Test()* function is called with the host part of the masked destination address and the trapdoor value in each masked address resolution cache table entry as parameters.

- ◇ In case that *Test()* returns 1 for a cache entry, the Ethernet source and destination addresses of the packet are updated, and the controller component orders the network node to send the packet to the destination endpoint.

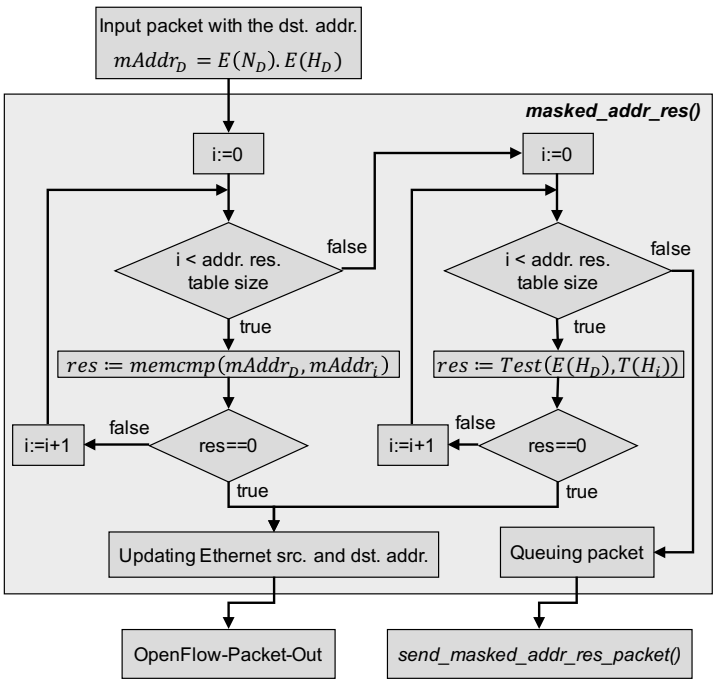


Figure 3.13: Flowchart for masked address resolution.

◇ If *Test()* returns 0 for all entires, the packet is put in a FIFO queue, and the controller component generates a masked address resolution packet, and it instructs the network node to broadcast the resolution packet in the local network by calling the function *send_masked_addr_res_packet()*.

After receiving the reply packet at the network node, the packet is sent to the controller component which calls the function *handle_masked_addr_res_packet()* on the object representing the network node. This function fetches the packet from the queue, and updates the Ethernet source and destination addresses of the packet. After that, the controller component tells the network node to send the packet to the destination endpoint.

3.3.2 Host side

On the host side, we have created a framework in Linux that implements a basic network stack with a socket-like interface in the user space. This network stack imitation, which we call *Blind Network Stack (BNS)*, realises the main functionalities of the transport (only UDP) and network layer (see Figure 3.14). In our implementation we have realised BNS as a C++ singleton class whose UML diagram is illustrated in Figure 3.15.

For the initialisation, a UNIX socket and a RAW socket is created in the function *init()*

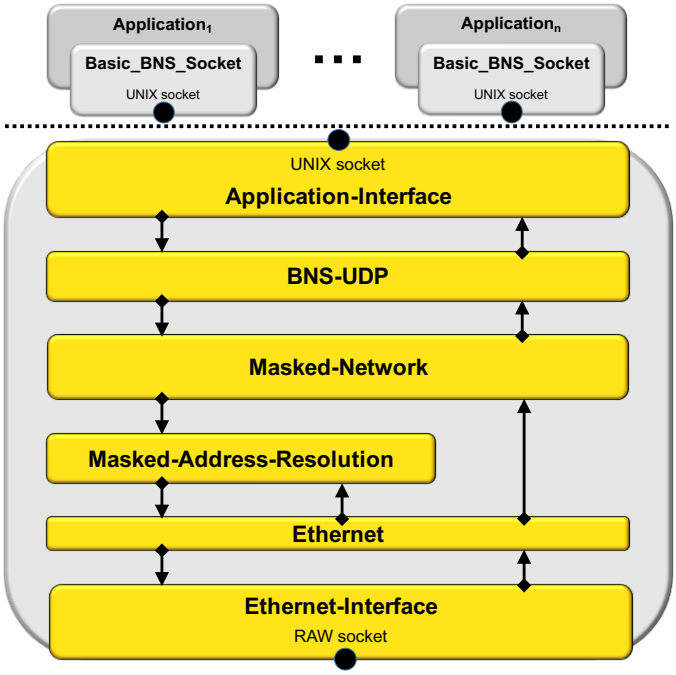


Figure 3.14: Blind Network Stack.

which takes the endpoint ID and an Ethernet interface index as parameters. Ethernet frames are sent and received by means of the RAW socket bound to the specified Ethernet interface. BNS gets user payloads from applications via the UNIX socket whose address is defined using the ID of the endpoint. Moreover, BNS generates a key pair and a trapdoor value for the ID of the endpoint, and encrypts the endpoint ID by means of PEKS.

After the initialisation, BNS broadcasts a request in the local network to which the endpoint is connected. The network node responsible for the local network responds to the request with its public key and network ID. After receiving the reply, BNS encrypts the ID of the network node with the public key contained in the reply message, and it thus creates its masked network address. After masking the network address, two threads are created and started. In the first one, BNS listens on the UNIX socket to get user payloads from applications, and in the second one, it listens on the RAW socket to receive Ethernet frames.

For the communication between BNS and an application, we have implemented an interface which we call *Basic_BNS_Socket*. This interface provides the functions *bns_socket_create()*, *bns_socket_send()*, and *bns_socket_receive()* similar to a conventional UDP socket in Linux. By calling the first function, the application creates a UNIX socket bound to an address which is identified using the number of the BNS-UDP port on which the application listens.

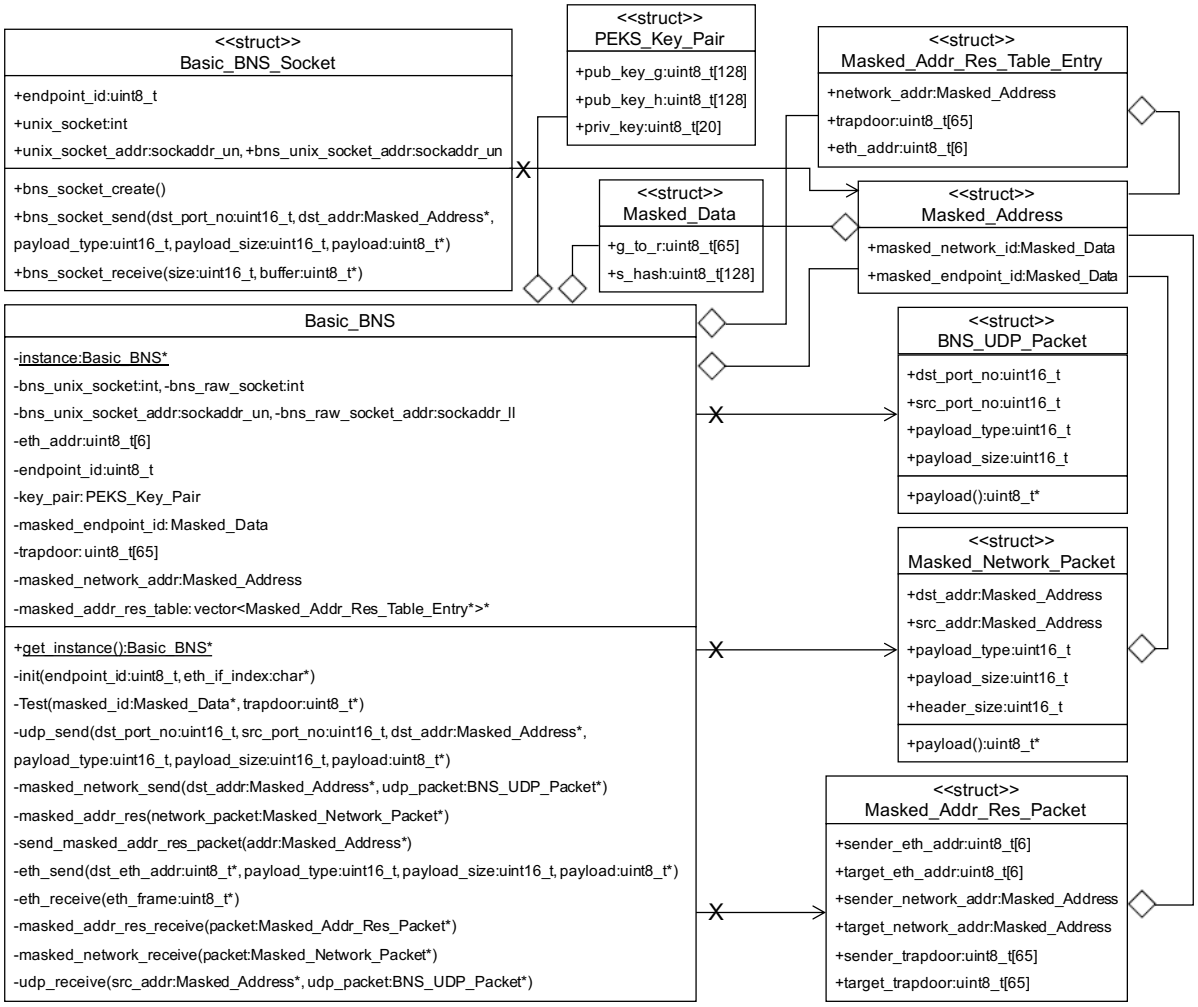


Figure 3.15: UML diagram of *Basic_BNS* and *Basic_BNS_Socket*.

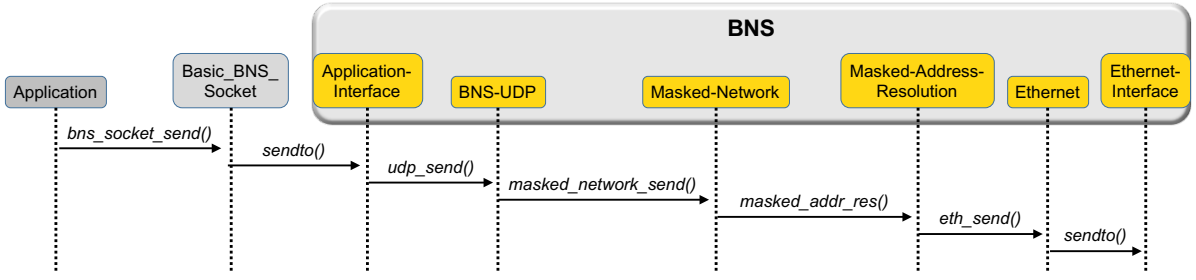


Figure 3.16: Message sequence chart for sending user payload.

In order to send an user payload, the following processing sequence is performed (see Figure 3.16):

1. After encrypting the destination network address by means of PEKS, an application calls the function *bns_socket_send()*. This function takes the destination BNS-UDP port number, the unmasked and masked destination network address, and the user payload as parameters which are sent to the address of the UNIX socket on the BNS side.

2. After receiving the parameters by BNS, they are delivered to the BNS-UDP module by calling the function *udp_send()*. This function generates a BNS-UDP datagram consisting of the source and destination BNS-UDP port number, the payload type and size, and the user payload itself.

3. Via the function *masked_network_send()*, the Masked-Network module takes the unmasked and masked destination network address and the BNS-UDP datagram to be sent. This module puts the masked network header in front of the datagram and hands the masked network datagram to the Masked-Address-Resolution module.

4. The Masked-Address-Resolution module takes the masked network datagram via the function *masked_addr_res()*. After resolving the masked destination network address to an Ethernet address, the masked network datagram is passed to the Ethernet module.

5. Via the function *eth_send()*, the Ethernet module takes the destination Ethernet address, the payload type and size, and the payload (the masked network datagram). This module puts the Ethernet header in front of the datagram, and eventually sends the Ethernet frame via the RAW socket.

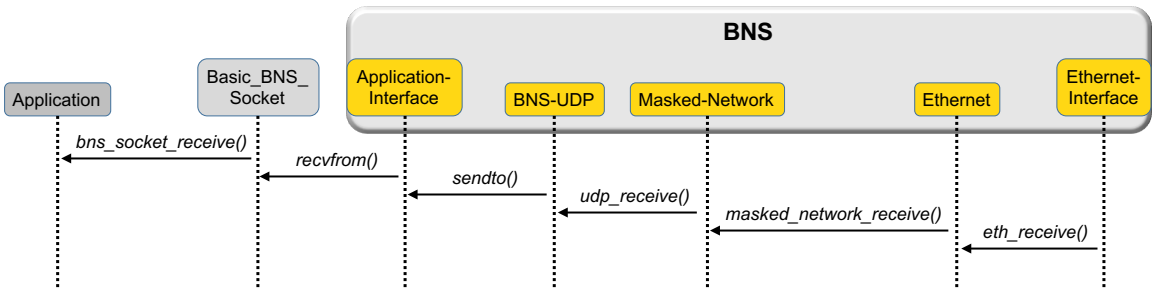


Figure 3.17: Message sequence chart for handling masked network packet.

An Ethernet frame arriving via the RAW socket is handled as follows (see Figure 3.17):

1. The frame is handed on the Ethernet module by calling the function *eth_receive()*. If the frame is broadcasted, or it is addressed to the Ethernet address of the interface via which the frame arrived, the Ethernet payload type of the frame is checked.

   ◇ In case of a masked address resolution packet, the packet is passed to the Masked-Address-Resolution module after removing the Ethernet header.

   ◇ If the value of the Ethernet payload type is equal to the type value of a masked network packet, the masked network datagram is delivered to the Masked-Network module.

2. Via the the function *masked_network_receive()*, the Masked-Network module takes the masked network datagram. After checking the masked destination network address of the packet by means of the function *Test()*, the BNS-UDP datagram and the masked source network address is given over to the BNS-UDP module.

3. Eventually, the BNS-UDP module receives the user payload, the source BNS-UDP port number, and the masked source network address via the function *udp_receive()* and sends them to the address of the UNIX socket on the application side. The address of the application UNIX socket is identified using the the destination BNS-UDP port number contained in the BNS-UDP header.

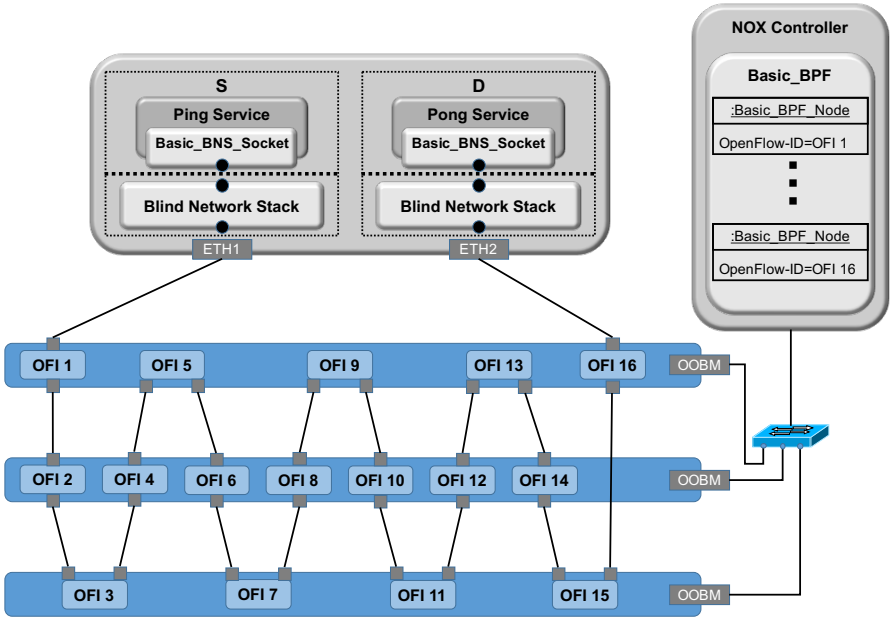


Figure 3.18: OpenFlow testbed for the basic BPF design.

3.4 Testbed

We have deployed the prototype implementation of the basic BPF design in a real hardware testbed to demonstrate its feasibility for practical deployment. Our testbed consists of two PCs and three HP 3800-24G-2SFP+ switches [Hew16] as shown in Figure 3.18.

The first PC has two network interfaces and an Intel Core2 Duo 3.33 GHz CPU. On this PC, we run two instances of BNS, which represent two endpoints. At the first endpoint, a ping service runs, while a pong service runs at the second endpoint. The extended NOX controller runs on the other PC with an Intel Core 2 Extreme 3.06 GHz CPU.

A HP 3800 switch can be configured in the virtualisation mode to create multiple HP OpenFlow instances. If the instances are configured in a port-based way, as it is the case in our configuration, they act as real OpenFlow switches. In our configuration, the switch ports are assigned to 16 OpenFlow instances that are interconnected with each other according to the testbed topology.

For the deployment of our implementation, we have decided on a linear topology consisting of 16 network nodes (see Figure 3.19). Thus, each node maintains 16 routing table entries. While the source endpoint S is connected to the first node, the destination endpoint D is connected to the last node so that a packet from S to D, or vice versa, takes 16 hops.

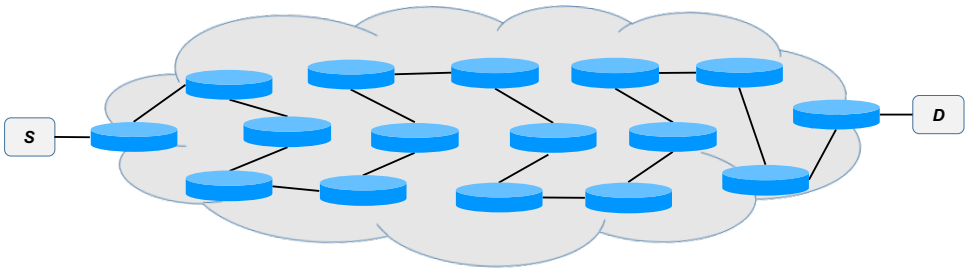


Figure 3.19: Testbed topology for the basic BPF design.

3.5 Evaluation

For the evaluation of our implementation, we first have benchmarked the cryptographic operations on the PC with an Intel Core2 Duo 3.33 GHz CPU (see Table 3.1). The first row in this table shows the execution time for generating a key pair and the sizes of the public and private key. In the second row, we can see the completion time for masking a network

or endpoint ID and the size of the masked ID, while the execution time for generating a trapdoor value and its size for a node or endpoint ID are shown in the third row. The last row shows the time required to preform the *Test* method for a masked ID and trapdoor value.

	Execution time in milliseconds	***Output size in bytes***
$KeyGen() \rightarrow (X_{pub}, X_{priv})$	4.37	***Public key:*** 256 ***Private key:*** 20
$X \rightarrow E(X)$	9.94	193
$X \rightarrow T(X)$	6.14	65
$Test(E(X), T(X))$	1.86	—

Table 3.1: Execution times and output sizes for the basic functionalities.

Table 3.2 presents the convergence times for the unmasked and masked routing table setup in the testbed topology described in Section 3.4, and the execution times for resolving an unmasked and a masked network address to a MAC address in case of an empty neighbour cache. In addition, we present the sizes of unmasked and masked structures implemented for the basic BPF design in Table 3.3. Furthermore, both tables show the increasing factors of sizes and execution times in the implementation of the basic BPF design in comparison with the unmasked packet forwarding.

	Time in milliseconds		***Increase***
	Unmasked	***Masked***	***by factor***
Routing table setup	739.12	52813.59	71.45
Address resolution	1.66	4.78	2.87

Table 3.2: Execution times for unmasked and masked routing and address resolution.

	Size in bytes		***Increase***
	Unmasked	***Masked***	***by factor***
Network address	2	386	193
Network packet header	10	778	77.8
Routing update entry	2	259	129.5
Address resolution packet	16	914	57.12
Routing table entry	12	268	22.33
Address resolution table entry	8	457	57.12

Table 3.3: Sizes of unmasked and masked structures.

The performance of our controller component and the blind network stack is evaluated in a ping-pong scenario in the testbed topology. By means of the conventional and blind packet forwarding, the endpoint S pings the endpoint D ten times for each case, while D responds by sending a pong packet back to S for every ping packet received. Figure 3.20 shows the packet round trip times for the unmasked and masked ping and pong packets. Thus, the average round trip time for the conventional packet forwarding is 39.77 milliseconds, while it is 888.15 milliseconds for the blind packet forwarding, representing an increase by factor 22.33. In principle, the performance can be boosted by integrating the blind network stack into the Linux kernel. Moreover, the choice of the controller hardware is crucial for the throughput.

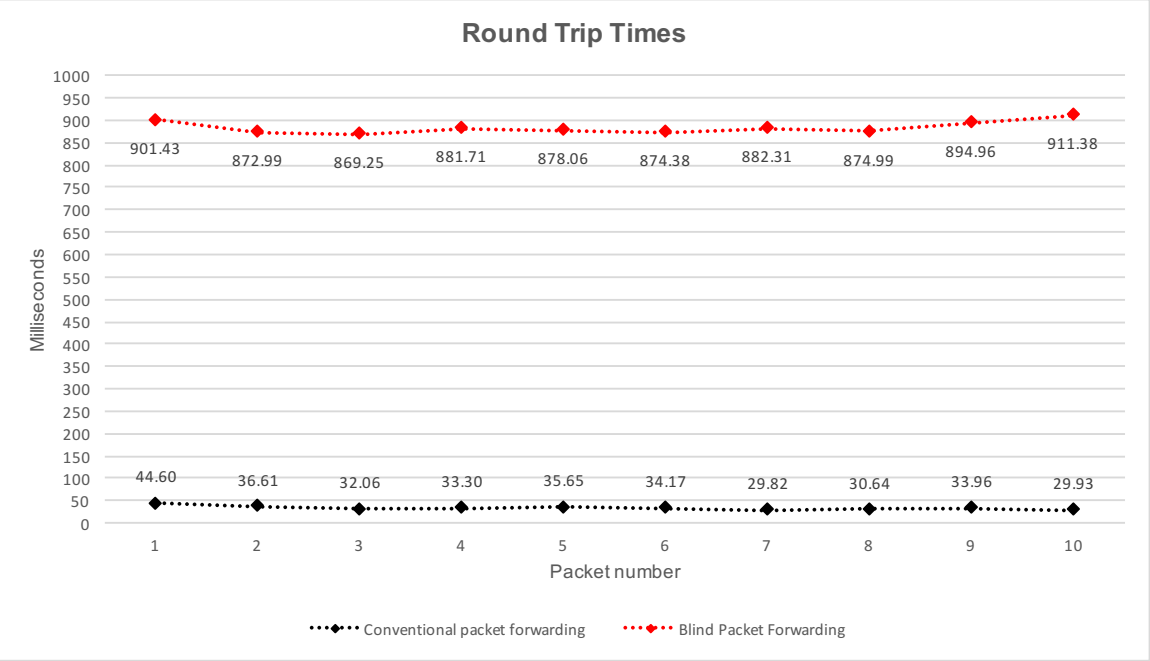


Figure 3.20: Packet round trip times.

The prototype implementation of the basic BPF design and its evaluation in the real hardware testbed proves the general feasibility of this approach and its practical deployment capability. However, the security provided by the basic BPF design introduces a certain amount of size and execution time overhead as is usually the case for every security mechanism. In comparison with the conventional packet forwarding, the basic BPF brings in an execution time overhead increase by a factor 32.21 with regard to the average RTT and Table 3.2, and a size overhead increase by a factor 85.82 with regard to Table 3.3 in average. This enormous amount of overhead leads us to state that the basic BPF design cannot satisfactorily be put into practice in its entirety for environments such as the Internet nowadays, at least not by default.

3.6 Conclusion

This chapter has presented a basic redesign of packet forwarding and the associated services – routing and address resolution – to blind ones which can still correctly process masked network addresses being based on a basic structure. To encrypt network addresses, we have used the PEKS algorithm. This algorithm is regarded as semantic-secure, and the encryption function of PEKS is not deterministic.

In this chapter, we have defined NAC which classifies only the source and destination endpoints as authorised entities having access to the cleartext addresses of packets transferred between both endpoints. In Section 3.2, we have discussed NAC and its deduced security properties – sender/recipient and relationship unlinkability. After analysing these properties in related works, it has been stated that the basic BPF design demonstrates the feasibility to simultaneously establish the packet forwarding service and to provide NAC. In addition, we have discussed BPF's sender/recipient and relationship unlinkability against different adversary models – local adversary, weak global adversary, and strong global adversary. Moreover, we have declared the effects of applying BPF in the current Internet architecture.

This chapter has also discussed the prototype implementation of the basic BPF design. For the implementation on the network side, we have utilised the SDN protocol OpenFlow 1.3 by expanding the OpenFlow controller NOX by a further component. On the host side, we have created a framework in Linux that implements a basic network stack with a socket-like interface in the user space. The prototype implementation of the basic BPF design has been deployed in a real hardware testbed to demonstrate its feasibility for practical deployment capability. For the evaluation of our implementation, we have benchmarked the cryptographic operations. In addition, we have compared the sizes of unmasked and masked structures, and the execution times for the unmasked and masked processes. Furthermore, the performance of our controller component and the blind network stack has been evaluated in a ping-pong scenario in a linear topology consisting of 16 network nodes.

In summary, we can state that the basic BPF design and its implementation demonstrates the general feasibility of our approach and its practical deployment capability, while introducing a considerable amount of overhead. Moreover, the basic BPF design is not suitable to be deployed in the current Internet architecture. These issues will be tackled in the next chapter.

Chapter 4

Towards an adequate design for BPF

In the previous chapter, we have presented a basic design for Blind Packet Forwarding (BPF). This basic BPF design demonstrates the feasibility to simultaneously establish the packet forwarding service and to provide network address confidentiality (NAC) and its unlinkability properties. However, this design is not suitable to be deployed in the current Internet architecture as discussed in Section 3.2.3. Moreover, the basic BPF design introduces a considerable amount of overhead (see Section 3.5).

In order to design a suitable architecture for BPF, the architecture has to be based on a hierarchical addressing structure, where it is well-defined, how many parts an endpoint address consists of, and which part of the address has to be encrypted with which public key. Moreover, the architecture has to be designed in a way that we do not need an additional infrastructure for exchanging and certifying the public keys of network nodes. Additionally, the architecture itself has to be hierarchically structured by design so that no supplementary process is required to aggregate and partition the networks if necessary.

While an IP address identifies an endpoint (identifier) and describes its network attachment point (locator) at the same time, the *Locator/Identifier (Loc/ID) Split* principle [MZF07] separates the locator functionality from the identifier. Thus, the network address of an endpoint consists of a Loc and an ID part. The ID part serves to locate the endpoint within a local network to which the endpoint is connected, while the Loc part specifies the location of the local network in the entire infrastructure, e.g., in the Internet. This principle is regarded as the de facto addressing standard for Future Network Architecture (FNA) and supports scalability, mobility and multihoming by design [Li11], [SG 12]. Currently, there exists a wealth of approaches relying on the Loc/ID Split principle as introduced in Section 2.2. In order to achieve the features for an adequate BPF design, we choose two suitable Loc/ID Split approaches and extend the basic BPF design by means of them in Sections 4.1 and 4.2. Each of

these extensions resolves several issues of the basic BPF design and introduces its own benefits.

In the prototype implementation of the basic BPF design, we have utilised OpenFlow and the controller NOX. There, masked packets are handled by the controller hop-by-hop which causes a considerable overhead. Therefore, another contribution of this chapter is the reduction of this overhead by leveraging *flow-based forwarding* in OpenFlow.

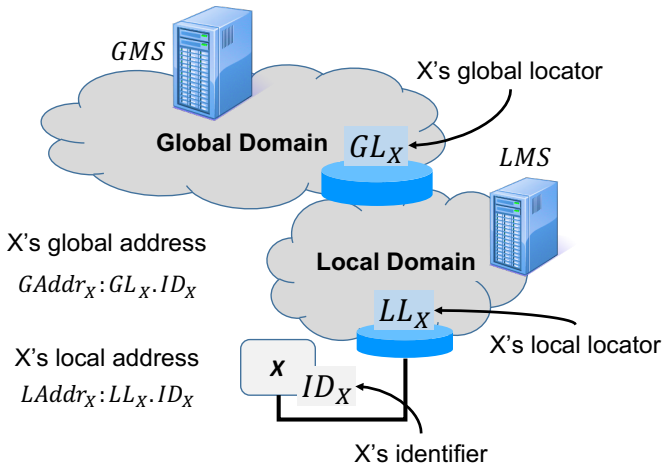


Figure 4.1: GLI-Split's architecture and addressing structure.

4.1 BPF using a Loc/ID Split approach

The *Global Locator, Local Locator, and Identifier Split (GLI-Split)* [MHK13] is one of the FNA approaches relying on the Loc/ID Split principle, which we have sketched in Section 2.2.1. This approach splits the functionality of IP addresses into global and local locators and identifiers. The architecture of GLI-Split divides the network into a global domain and multiple local domains connected with each other through the global domain (see Figure 4.1). GLI-Split separates global and local routing and performs core routing on global locators and edge routing on local locators. The global address of an endpoint consists of its global locator and identifier, while its local address is composed of its local locator and identifier. In GLI-Split, gateways (nodes at the border of a local domain and the global domain) substitute global and local locators with each other for incoming and outgoing packets. In a local network, the destination identifier of a packet is resolved to the MAC address of the destination endpoint, and the packet is forwarded to this MAC address.

The mapping system of GLI-Split consists of a *Global Mapping System (GMS)* and a *Local*

Mapping System (LMS) in each local domain. The GMS maps the identifier of an endpoint to a global locator, while the LMS in a local domain maintains the mapping of the identifier to a local locator. Here, the LMS in a local domain is under the control of the domain provider, while the GMS can be maintained by the local domain providers in a distributed manner. An endpoint newly connected to a local network registers the mapping of its identifier to its global and local locator at the GMS and LMS responsible for the local domain in which the endpoint resides. Before sending a packet, a source endpoint resolves the actual locator of the destination endpoint by querying the mapping system.

We adopt the architecture and addressing structure of GLI-Split and extend our BPF design to the *Blind Packet Forwarding in Global Locator, Local Locator, and Identifier Split (BPF-GLI)*. In this BPF extension, we construct threes modes, namely *semi-blind mode*, *fully blind mode*, and *alternately blind mode*, in the following sections. For identifier resolution, mapping registration, lookup, and packet delivery, BPF-GLI performs the same processes as defined in GLI-Split. In Section 4.1.4, we first discuss the architectural features of BPF-GLI and afterwards analyse the NAC levels and their unlinkability properties provided by the blindness modes of BPF-GLI. Section 4.1.5 presents the implementation of BPF-GLI and leveraging of the flow-based forwarding in OpenFlow. The deployment of our implementation in a real hardware testbed is given in Section 4.1.6. Section 4.1.7 evaluates the implementation of BPF-GLI.

For the construction, we assume that all network nodes, communicating endpoints, and mapping systems have already generated a key pair using PEKS. Moreover, it is assumed that the public keys of the communicating endpoints are already exchanged and bound to their owners. Furthermore, we make the assumption that the communicating endpoints have already found the identifiers of each other, e.g., via DNS.

4.1.1 Semi-blind packet forwarding

In this mode of BPF-GLI, we only mask identifiers, while global and local locators are handled in cleartext. Thus, the semi-masked global and local addresses of the endpoint X with the identifier ID_X are

$$smGAddr_X : GL_X.E(ID_X) \text{ and} \tag{4.1}$$

$$smLAddr_X : LL_X.E(ID_X), \text{ where} \tag{4.2}$$

- $E(ID_X) = PEKS(X_{pub}, ID_X)$ is X's masked identifier encrypted with its own public key X_{pub}.

- GL_X is the global locator of a gateway node responsible for the local domain in which X resides.
- LL_X is the local locator of the edge node responsible for the local network to which X is connected.

In case of intra-domain communication, the semi-masked address fields in a packet from the source endpoint S to the destination endpoint D consist of

$$smLAddr_D \mid smLAddr_S \mid C(ID_S). \tag{4.3}$$

If the communicating endpoints reside in different local domains, the semi-masked address fields consist of

$$\begin{aligned} &smGAddr_D \mid smLAddr_S \mid C(ID_S) \text{ in the source local domain,} \\ &smGAddr_D \mid smGAddr_S \mid C(ID_S) \text{ in the global domain, and} \\ &smLAddr_D \mid smGAddr_S \mid C(ID_S) \text{ in the destination local domain.} \end{aligned} \tag{4.4}$$

Here, the semi-masked global and local addresses of the destination and source endpoints are generated according to equations 4.1 and 4.2. Moreover, $C(ID_S)$ is the ciphertext generated by conventionally encrypting (e.g., with RSA) ID_S with the corresponding public key of D. In case of intra-domain communication, a packet from S to D contains their semi-masked local addresses. If both endpoints reside in different local domains, the address fields in the packet comprise the semi-masked global or local addresses of the endpoints according to the domain within which the packet is currently forwarded.

For semi-masked packet generation, the source endpoint encrypts only the identifier part of the destination address with the public key of the destination endpoint by using PEKS. In the same way, the source endpoint encrypts only the identifier part of its own address with its own public key. In case that the packet cannot be forwarded to the destination endpoint, the semi-masked address of the source endpoint serves as the destination address of a corresponding destination unreachable message. For sending a packet back to the source endpoint, the destination endpoint can use the second address field as the destination address of the packet. To get the identifier of the source endpoint in cleartext, the destination endpoint decrypts the ciphertext in the third address field. The source and destination endpoints can cache encrypted identifiers so that they do not have to encrypt the identifiers for each packet.

Since only identifiers are masked in this mode, a conventional (unmasked) routing table setup is established in local domains and in the global domain, e.g., by means of the Distance Vector Routing algorithm. Here, the routing table of a node in the global domain contains the

global locators (instead of the network addresses) of the other nodes, while a node in a local domain holds the local locators of the other local nodes in its routing table. Moreover, each node in a local domain maintains at least one routing table entry defining the default route to a gateway node responsible for the local domain. Furthermore, the gateway nodes maintains two tables, namely a global routing table and a local routing table. While the routing can be used as it is, the identifier resolution, mapping registration and lookup, however, have to be redesigned so that they can still correctly process masked identifiers.

4.1.1.1 Masked identifier resolution

In a local network, the masked destination identifier of a packet is resolved to the MAC address of the destination endpoint and the packet is forwarded as proposed in Section 3.1.4. For the host X with the identifier ID_X, the entry in a masked identifier resolution cache table consists of

$$[E(ID_X), T(ID_X), MAC_X], \text{ where}$$

- $E(ID_X) = PEKS(X_{pub}, ID_X)$ is the masked identifier of the host X.
- $T(ID_X) = Trapdoor(X_{priv}, ID_X)$ is X's trapdoor value for its identifier.
- MAC_X is the MAC address of the host X in cleartext.

In case that some entries in the neighbour cache already exist, the resolving of X's masked identifier $E(ID_X)$ occurs by performing

$$Test(E(ID_X), T(ID_i))$$

for each cache table entry i. The masked identifier is then resolved to the MAC address in the entry for which $Test()$ returns 1. Performing a binary comparison of the masked identifier with the masked identifiers in the cache table before executing $Test()$ can speed up the masked identifier resolution. Since however the byte value of $E(ID_X)$ is not constant, it is possible that no entry can be found in this way. In this case, the masked identifier is resolved as described above.

In case that no entry is found for X in that way, or the cache table is empty, the host D aiming to resolve the masked identifier $E(ID_X)$ to a MAC address broadcasts a request message consisting of the masked identifier which has to be resolved, its own masked identifier, trapdoor value, and MAC address (see Figure 4.2):

$$id\_res\_req\{E(ID_X), (E(ID_D), T(ID_D)), MAC_D\}.$$

After receiving the request message, each host X_j performs

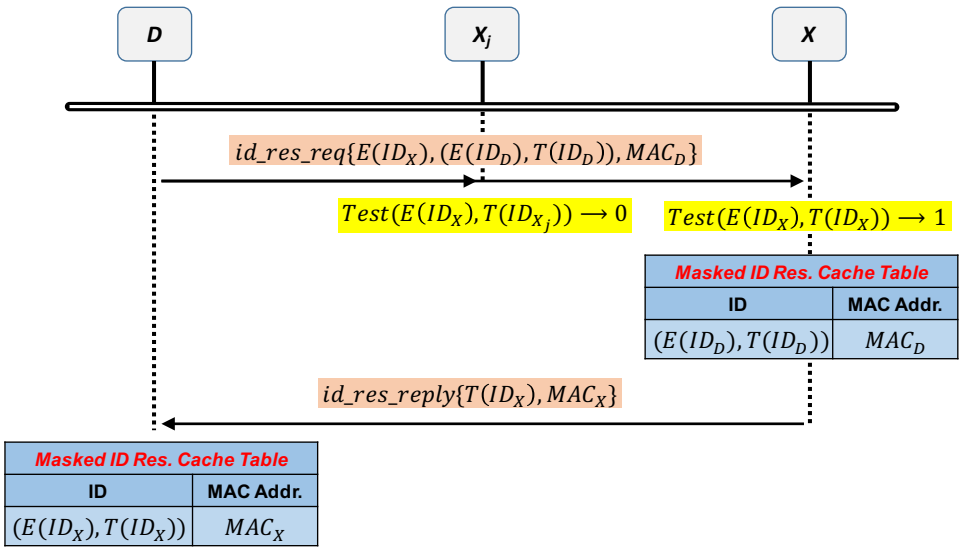


Figure 4.2: Masked identifier resolution.

$$Test(E(ID_X), T(ID_{X_j}))$$

in order to determine, which host is addressed with the request message. Hosts, at which $Test()$ returns 0, drop the request message, while the host X performs

$$Test(E(ID_D), T(ID_i))$$

for each entry i in its cache table. In this way, the host X finds out whether it already keeps a cache entry for the host D.

- ◇ If $Test()$ returns 1 for an entry, i.e. an entry already exists for the host D, the entry is updated with $E(ID_D)$ and MAC_D. In this way, the byte value of D's masked identifier is also updated.

- ◇ If $Test()$ returns 0 for all entries, i.e. no entry exists for the host D so far, a new entry is created with the values from the request message.

Then the host X sends a reply message containing its own trapdoor value, and MAC address

$$id\_res\_reply\{T(ID_X), MAC_X\}$$

to the host D. After receiving the reply message, the host D first creates a new entry with X's masked identifier, trapdoor and MAC address, and then it resolves the masked identifier to the MAC address.

4.1.1.2 Semi-blind mapping system

BPF-GLI in the semi-blind mode adopts the mapping system architecture of GLI-Split and only masks endpoint identifiers in the mapping tables of the GMS and $LMSs$. The GMS maintains a table which holds the semi-masked global mappings of endpoints. A *semi-masked global mapping table entry (sm-global-MTE)* for the endpoint X consists of the masked identifier, trapdoor and at least one cleartext global locator[1] of X:

$$\textit{sm-global-MTE}_X : [(E(ID_X), T(ID_X)), GL_X].$$

In the LMS responsible for the local domain in which the endpoint X resides, a *semi-masked local mapping table entry (sm-local-MTE)* for the endpoint X consists of X's masked identifier, trapdoor and local locator:

$$\textit{sm-local-MTE}_X : [(E(ID_X), T(ID_X)), LL_X].$$

Moreover, the LMS keeps the global locators of gateways connecting the local domain with the global domain and the semi-masked address of the GMS. If the mapping registration and lookup traffic does not have to be masked, the LMS can only hold GMS's cleartext address. Since the LMS and the gateways are under the control of the local domain provider, the LMS can be simply configured with the global locators.

As proposed in GLI-Split, an enhanced DHCP server in a local network can maintain locators and the addresses of the GMS and the LMS responsible for the local domain to which the local network belongs. Thus, the DHCP server holds the cleartext local locator of the edge node responsible for the local network, and the cleartext global locators of gateways connecting the global domain with the local domain in which the local network resides. In contrast to GLI-Split, the DHCP server in BPF-GLI keeps the semi-masked addresses of the GMS and LMS, if the mapping registration and lookup traffic also has to be masked. Otherwise, the DHCP server can simply maintain their unmasked addresses, just like in GLI-Split. The local network provider can get the above-quoted values from the local domain provider and configure the DHCP server with them.

4.1.1.3 Semi-masked mapping registration

An endpoint D, which is newly connected to a local network, has to register the mapping of its masked identifier to its global and local locator at the GMS and LMS_D (mapping system in the local domain to which the local network belongs). For that, the new endpoint D encrypts its identifier ID_D with its public key and generates the trapdoor $T(ID_D)$ for its identifier

[1] Because of multihoming possibility

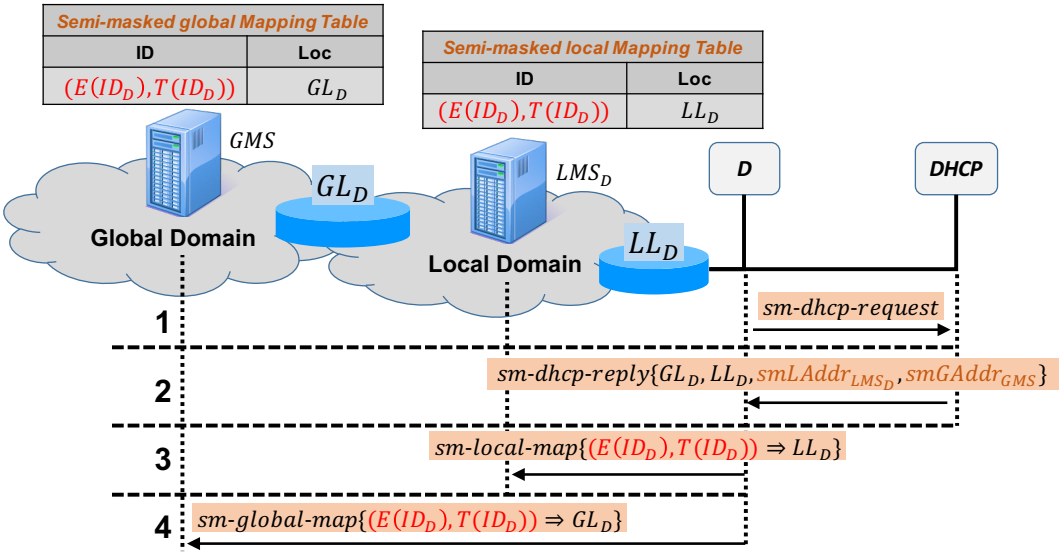


Figure 4.3: Semi-masked mapping registration in BPF-GLI.

with its private key by means of PEKS, and the same registration steps are performed as defined in GLI-Split (see Figure 4.3):

1. The endpoint D asks for a local and global locator, and the addresses of the LMS_D and GMS by sending a request to the DHCP server in the local network to which the endpoint is connected.

2. For an incoming request, the DHCP server responds with the local locator of the edge node responsible for the local network, and with the global locator of a gateway responsible for the local domain in which the endpoint is currently resides. In addition, the reply message contains the semi-masked addresses of the LMS_D and GMS:

$$sm\text{-}dhcp\text{-}reply\{LL_D, GL_D, smLAddr_{LMS_D}, smGAddr_{GMS}\}.$$

3. The endpoint D compares the local locator from the reply message with its old local locator. If both locators are different, or the endpoint did not have any local locator so far, the endpoint caches the values from the reply message and sends its semi-masked local mapping

$$sm\text{-}local\text{-}map\{(E(ID_D), T(ID_D)) \Rightarrow LL_D\} \text{ to the } LMS_D.$$

For the incoming semi-masked mapping, the LMS_D performs $Test(E(ID_i), T(ID_D))$ for each entry i.

- ◇ If $Test()$ returns 1 for an entry, i.e. a mapping for D already exists, the entry is updated with the local locator and masked identifier from the incoming mapping. Thus, the ciphertext for the identifier is updated with the ciphertext newly generated by the registering endpoint.
- ◇ If $Test()$ returns 0 for all entries, i.e. no entry for D exists so far, a new entry is created with the values from the incoming semi-masked mapping.

4. In case that the endpoint has got a new global locator or did not have any global locator so far, the endpoint registers itself at the GMS by sending its semi-masked global mapping

$$\textit{sm-global-map}\{(E(ID_D), T(ID_D)) \Rightarrow GL_D\} \text{ to the } GMS.$$

The GMS first checks whether it already maintains a mapping entry for the endpoint by performing $Test()$ with the trapdoor in the incoming mapping and the masked identifier in each table entry as parameters. In case that an entry already exists, the entry is updated with the global locator and masked identifier from the incoming mapping. Otherwise, the GMS creates a new entry.

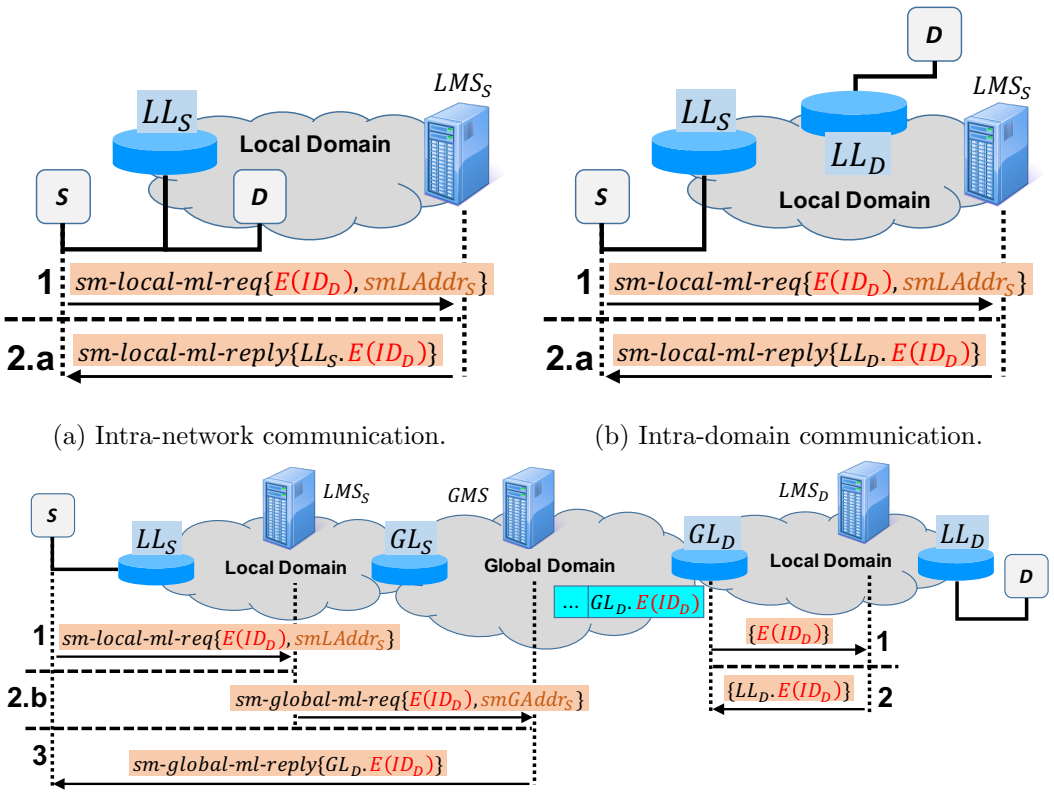
Here, the endpoint uses its newly configured semi-masked local and global addresses as the source addresses of the network packets carrying the semi-masked local and global mappings. Moreover, it is also possible that the DHCP server responds with the unmasked addresses of the LMS_D and GMS, and the endpoint uses them as the destination addresses of the mapping registration messages, if the destinations of the mapping registration traffic do not have to be masked.

4.1.1.4 Semi-masked mapping lookup

Before sending a semi-masked packet from the source endpoint S to the destination endpoint D, S has to resolve D's actual locator. By means of PEKS, S first encrypts the destination identifier ID_D with D's public key. After that, the same mapping lookup steps are performed as defined in GLI-Split (see Figure 4.4):

1. The source endpoint S generates a request message for $E(ID_D)$ and sends it to the mapping system at which S is registered, namely to the LMS_S. Here, S sets its own semi-masked local address as the source address of the semi-masked local mapping lookup request:

$$\textit{sm-local-ml-req}\{E(ID_D), smLAddr_S\}.$$



(a) Intra-network communication.

(b) Intra-domain communication.

(c) Inter-domain communication.

Figure 4.4: Semi-masked mapping lookup in BPF-GLI.

2. For the incoming request message, the LMS_S performs $Test(E(ID_D), T(ID_i))$ for each entry i.

   (a) If $Test()$ returns 1 for an entry, i.e. S and D are connected to the same edge node or to different edge nodes in the same local domain, the LMS_S responds with D's semi-masked local address (see Figures 4.4a and 4.4b).

   (b) If $Test()$ returns 0 for all entries, the LMS_S replaces the local locator from the semi-masked local address of the requesting endpoint with the global locator of a gateway responsible for the local domain and forwards the semi-masked global mapping lookup request to the GMS (see Figure 4.4c):

   $$sm\text{-}global\text{-}ml\text{-}req\{E(ID_D), smGAddr_S\}$$

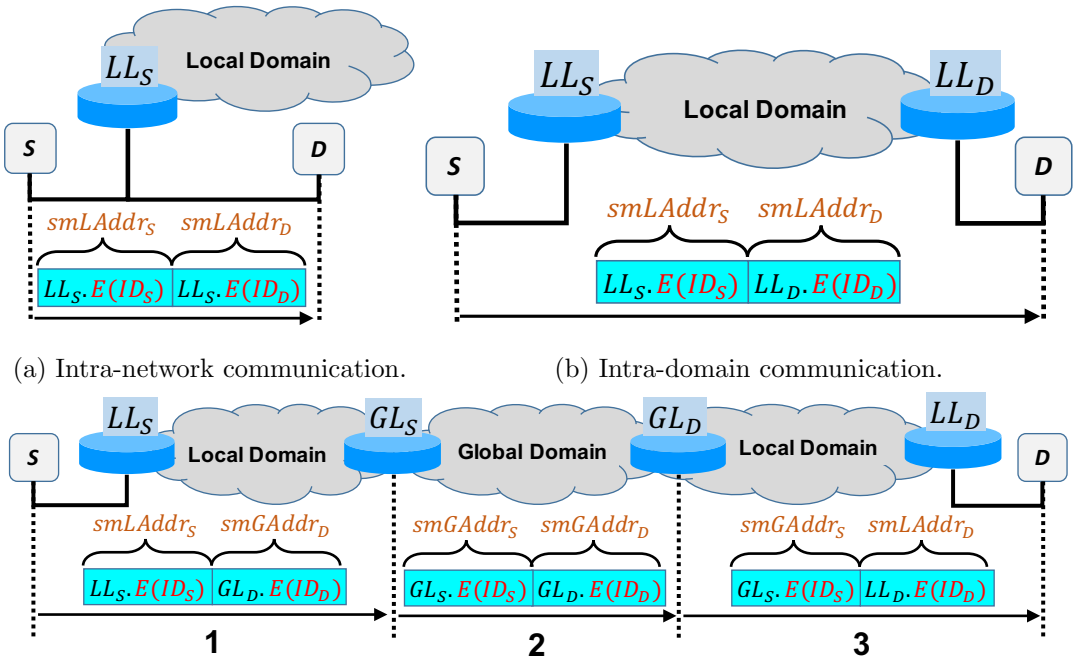
3. For the forwarded request message, the GMS performs $Test(E(ID_D), T(ID_i))$ for each mapping table entry i, and it thus determines D's actual global locator. Finally, the GMS responds with D's semi-masked global address.

The endpoint S can cache the locator so that it does not have to perform a mapping lookup for each packet.

For a packet, which is coming from the global domain and contains GL_D as its destination global locator, the gateway GL_D has to resolve D's masked identifier to its local locator (see Figure 4.4c):

1. The gateway node queries the mapping system responsible for the local domain, namely the LMS_D, for D's actual local locator by sending a request message containing $E(ID_D)$.
2. For the request message, the LMS_D performs $Test()$ with D's masked identifier and the trapdoor in each mapping table entry, and it responds with D's semi-masked local address containing the local locator mapped in the entry for which $Test()$ returns 1.

The gateway GL_D can temporally cache the locator for further packets addressed to the endpoint D.



(a) Intra-network communication. (b) Intra-domain communication.

(c) Inter-domain communication.

Figure 4.5: Semi-masked packet delivery in BPF-GLI.

4.1.1.5 Semi-masked packet delivery

The endpoint S wants to send a packet to the endpoint D. We assume that the endpoints have already registered their semi-masked local and global mappings at the mapping systems

as explained in Section 4.1.1.3. First, S encrypts D's identifier with the public key D_{pub} by means of PEKS. After that, S resolves D's masked identifier to its actual locator as described in Section 4.1.1.4.

- ◇ In case of intra-domain communication, S gets the semi-masked local address $smLAddr_D$, and it thus generates the semi-masked packet $< smLAddr_D \mid smLAddr_S >$.
  - ◇ If both endpoints possess the same local locator, i.e. they are connected to the same local network (see Figure 4.5a), the masked destination identifier is resolved to the MAC address of the destination endpoint as described in Section 4.1.1.1. Finally, the packet is forwarded to this MAC address.
  - ◇ Otherwise, i.e. the endpoints are connected to different edge nodes in the same local domain (see Figure 4.5b), the semi-masked packet is locally forwarded to the edge node responsible for the local network to which D is connected. After receiving the packet, the edge node performs the resolution for the masked destination identifier, and it forwards the packet to the destination endpoint.
- ◇ If the endpoints reside in different local domains, the GMS responds with D's semi-masked global address $smGAddr_D$, and the source endpoint generates the semi-masked packet $< smGAddr_D \mid smLAddr_S >$. The packet is forwarded in three steps as defined in GLI-Split (see Figure 4.5c):
  1. The source endpoint S sends the packet to its edge node LL_S, and the packet is forwarded to the gateway node GL_S by using the default route.
  2. Upon receiving the packet, GL_S substitutes $smLAddr_S$ with $smGAddr_S$ by replacing LL_S with GL_S, and it forwards the packet $< smGAddr_D \mid smGAddr_S >$ within the global domain.
  3. When the packet arrives at the gateway node GL_D, it gets the local locator mapped to D's masked identifier from the LMS responsible for the local domain as described in Section 4.1.1.4. After substituting $smGAddr_D$ by $smLAddr_D$, the gateway node locally forwards the packet $< smLAddr_D \mid smGAddr_S >$. Eventually, the packet arrives at the edge node LL_D, which resolves D's masked identifier to its MAC address. Finally, the packet is delivered to D's MAC address.

4.1.2 Fully blind packet forwarding

In order to achieve a full blindness, we mask both locators and identifiers in this mode. Thus, the fully masked global and local addresses of the endpoint X with the identifier ID_X are

$$fmGAddr_X : E(GL_X).E(ID_X) \text{ and} \tag{4.5}$$

$$fmLAddr_X : E(LL_X).E(ID_X), \text{ where} \tag{4.6}$$

- $E(ID_X) = PEKS(X_{pub}, ID_X)$ is the masked identifier of the endpoint, which is encrypted with its public key X_{pub}.
- $E(GL_X) = PEKS(GL_{X_{pub}}, GL_X)$ is X's masked global locator generated with the public key of the gateway node responsible for the local domain in which X resides.
- $E(LL_X) = PEKS(LL_{X_{pub}}, LL_X)$ is X's masked local locator generated with the public key of the edge node responsible for the local network to which X is connected.

If the source endpoint S and the destination endpoint D are connected to edge nodes in the same local domain, the fully masked address fields in a packet from S to D consist of

$$fmLAddr_D \mid fmLAddr_S \mid C(ID_S). \tag{4.7}$$

In case of inter-domain communication, the fully masked address fields consist of

$$\begin{aligned} &fmGAddr_D \mid fmLAddr_S \mid C(ID_S) \text{ in the source local domain,} \\ &fmGAddr_D \mid fmGAddr_S \mid C(ID_S) \text{ in the global domain, and} \\ &fmLAddr_D \mid fmGAddr_S \mid C(ID_S) \text{ in the destination local domain.} \end{aligned} \tag{4.8}$$

While the global and local addresses of the destination and source endpoints are fully masked according to equations 4.5 and 4.6, the source identifier of the packet is additionally encrypted (e.g., by means of RSA) with the corresponding public key of D.

The second and third address fields serve the same purposes as described in semi-blind packet forwarding. In case of intra-domain communication, the address fields contain the fully masked local addresses of the communicating endpoints. Moreover, a packet transferred via the global domain comprises the fully masked global or local addresses of the endpoints. Here, it is crucial for the address fields, within which domain the packet is currently forwarded, as is the case in the semi-blind mode.

Although the locators are handled in this mode in encrypted form, the source endpoint itself has to encrypt only the identifier of the destination endpoint and its own identifier for fully masked packet generation. Thus, the source endpoint needs only the public key of the destination endpoint, just like in semi-blind packet forwarding. In order to achieve that, the mapping registration and lookup have to be designed accordingly. Moreover, the routing tables of nodes in the global domain and in local domains have to be restructured in a way that the nodes can still correctly forward packets with masked locators.

4.1.2.1 Masked routing

For a masked routing table setup, we redesign the Distance Vector Routing algorithm to a blind one in the same manner as proposed in Section 3.1.3. The masked routing table entry for the node N with the locator Loc_N is

$$mRTE_N : [(E(Loc_N), T(Loc_N)), P_N, D_N], \text{ where}$$

- $E(Loc_N) = PEKS(N_{pub}, Loc_N)$ is N's masked locator encrypted with its own public key N_{pub}.
- $T(Loc_N) = Trapdoor(N_{priv}, Loc_N)$ is the trapdoor value for N's locator, which is generated with its own private key N_{priv}.
- P_N is the number of the port via which the network node N can be reached.
- D_N is the distance to the network node N.

The masked routing update and the link failure are handled in the same manner as described in the basic BPF design. However, the masked global and local routing is performed separately so that the changes in a local domain do not affect the global domain, and vice versa. Thus, a gateway node maintains two masked routing tables. In the first table, the gateway node holds masked routing information for the nodes in the global domain. Moreover, the second table consists of the entries for the nodes in the local domain which is connected with the global domain by the gateway node. Furthermore, each node in a local domain maintains at least one routing table entry defining the default route to a gateway node responsible for the local domain.

4.1.2.2 Fully blind mapping system

In this mode, BPF-GLI masks both identifiers and locators in the mapping tables of the GMS and $LMSs$. As in the semi-blind mode, BPF-GLI in the fully blind mode adopts the mapping system architecture of GLI-Split. In the GMS, a *fully masked global mapping table entry (fm-global-MTE)* for the endpoint X consists of its masked identifier, trapdoor and masked global locator:

$$fm\text{-}global\text{-}MTE_X : [(E(ID_X), T(ID_X)), E(GL_X)].$$

In each local domain, a LMS maintains the mappings of masked identifiers to masked local locators for the endpoints residing in the local domain. In the LMS responsible for the local domain in which the endpoint X is located, a *fully masked local mapping table entry (fm-local-MTE)* for the endpoint X consists of the masked identifier, trapdoor and masked local locator of X:

$$fm\text{-}local\text{-}MTE_X : [(E(ID_X), T(ID_X)), E(LL_X)].$$

Additionally, the LMS holds the global locators and PEKS-key pairs of gateway nodes connecting the local domain with the global domain as well as the fully masked address of the GMS. Since the gateways and the local mapping system are under the control of the local domain provider, an additional system is not needed to certify the keys.

As in the semi-blind mode, an enhanced DHCP server in a local network keeps some values. Here, the server maintains the local locator and PEKS-key pair of the edge node. Moreover, the server obtains the masked global locators and trapdoors of the gateways responsible for the local domain as well as the fully masked addresses of the GMS and LMS from the local domain provider. If the mapping registration and lookup traffic does not have to be masked, the server just has their unmasked addresses.

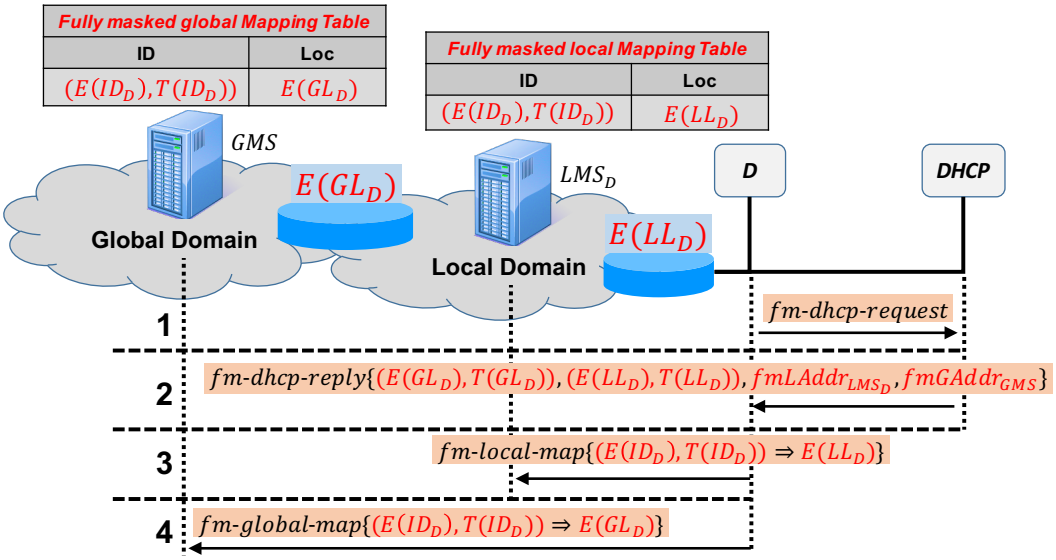


Figure 4.6: Fully masked mapping registration in BPF-GLI.

4.1.2.3 Fully masked mapping registration

If the endpoint D is newly added to a local network, the mapping of its masked identifier to its masked global and local locator has to be registered at the GMS and at the mapping system responsible for the local domain to which the local network belongs (LMS_D). After encrypting the identifier of the endpoint and generating the trapdoor for the identifier by means of PEKS, the same steps are performed as in GLI-Split (see Figure 4.6):

1. The new endpoint queries the DHCP server in the local network in order to configure

itself with a masked global and local locator as well as the fully masked addresses of the GMS and LMS_D.

2. For a incoming request, the DHCP server responds with the masked local locator and trapdoor of the edge node as well as the masked global locator and trapdoor of a gateway responsible for the local domain. Moreover, the reply message contains the fully masked addresses of the LMS_D and GMS:

$$\textit{fm-dhcp-reply}\{(E(LL_D), T(LL_D)), (E(GL_D), T(GL_D)), fmLAddr_{LMS_D}, fmGAddr_{GMS}\}.$$

3. The endpoint performs $Test(E(\textit{old-LL}), T(LL_D))$. Here, $E(\textit{old-LL})$ is the PEKS-ciphertext for the old local locator of the endpoint. If $Test()$ returns 0 (i.e., both local locators are different), or the endpoint did not have any masked local locator so far, the endpoint caches the values from the reply message and sends its fully masked local mapping

$$\textit{fm-local-map}\{(E(ID_D), T(ID_D)) \Rightarrow E(LL_D)\} \text{ to the } LMS_D.$$

For the incoming fully masked mapping, the LMS_D performs $Test(E(ID_i), T(ID_D))$ for each mapping table entry i.

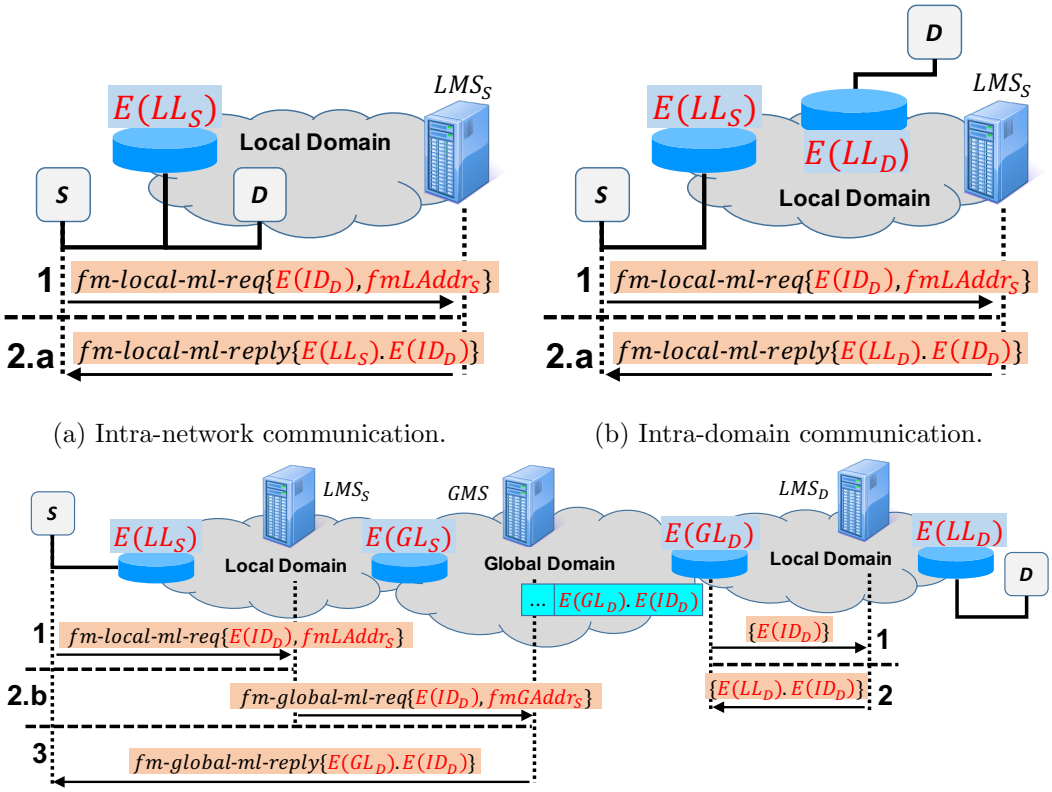
   - ◇ If $Test()$ returns 1 for an entry, the masked identifier and local locator in the entry are updated with the values from the mapping message. In this way, the byte values of the masked identifier and local locator are also updated.
   - ◇ If $Test()$ returns 0 for all entries, a new entry is created with values from the incoming fully masked mapping.

4. By performing $Test()$ with the PEKS-ciphertext for the old global locator and the trapdoor for the global locator from the DHPC reply message as parameters, the endpoint checks whether it has got a new global locator. If this is the case, or the endpoint did not have any masked global locator so far, the endpoint sends its fully masked global mapping

$$\textit{fm-global-map}\{(E(ID_D), T(ID_D)) \Rightarrow E(GL_D)\} \text{ to the } GMS.$$

The GMS proceeds analogously and updates the already existing entry or creates a new one for the endpoint.

For the registration messages, the endpoint uses its newly configured fully masked global and local addresses. Moreover, if the destinations of the registration traffic do not have to be masked, the endpoint gets the unmasked addresses of the mapping systems and uses them as the destination addresses of the network packets carrying the registration messages.



(a) Intra-network communication.

(b) Intra-domain communication.

(c) Inter-domain communication.

Figure 4.7: Fully masked mapping lookup in BPF-GLI.

4.1.2.4 Fully masked mapping lookup

In this mode of the mapping lookup, a source endpoint has to resolve the masked destination identifier to a masked locator before sending a fully masked packet. For that, the same mapping lookup steps are performed as in GLI-Split (see Figure 4.7):

1. The source endpoint S first queries the local mapping system LMS_S for a masked local locator of the destination endpoint D by sending a request message containing the masked destination identifier $E(ID_D)$. Here, the requesting endpoint sets its own fully masked local address as the source address of the fully masked local mapping lookup request:

$$fm\text{-}local\text{-}ml\text{-}req\{E(ID_D), fmLAddr_S\}.$$

2. For the incoming request, the LMS_S performs $Test()$ with the masked identifier from the message and the trapdoor in each fully masked mapping table entry as parameters.

   (a) If the source and destination endpoints reside in the same local domain, $Test()$

returns 1 for an entry, and the LMS_S responds with D's fully masked local address (see Figures 4.7a and 4.7b).

(b) Otherwise (i.e, $Test()$ returns 0 for all entries), the LMS_S replaces the masked local locator from the fully masked local address of the requesting endpoint with the masked global locator of a gateway responsible for the local domain. After that, the LMS_S sends the fully masked global mapping lookup request to the GMS (see Figure 4.7c):

$$fm\text{-}global\text{-}ml\text{-}req\{E(ID_D), fmGAddr_S\}.$$

3. For the incoming request, the GMS proceeds analogously in order to determine the fully masked mapping table entry for the requested identifier, and it thus responds with D's fully masked global address.

The source endpoint can cache the masked locator so that it does not have to perform a fully masked mapping lookup for each packet.

If the gateway node GL_D receives a packet, which comes from the global domain and contains GL_D's masked global locator as its destination global locator, the gateway node has to resolve D's fully masked local address (see Figure 4.7c):

1. The gateway node sends a request message to the local mapping system responsible for the local domain (LMS_D). This request contains the masked destination identifier of the packet.

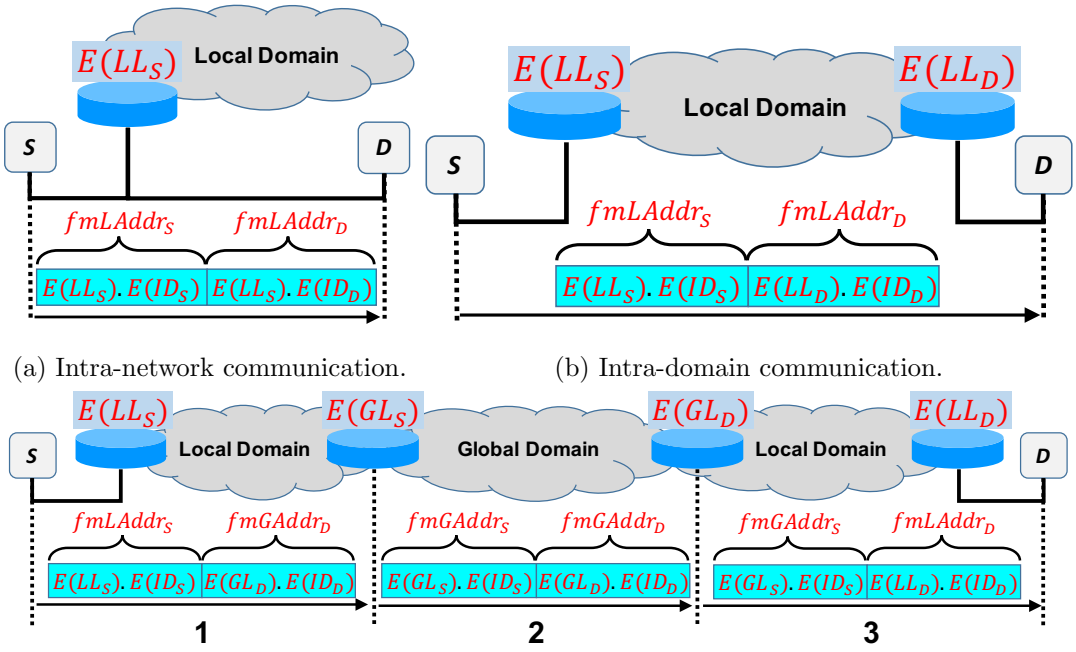
2. For the request, the LMS_D determines D's masked local locator in the same manner as described above. Finally, the mapping system responds with D's fully masked address.

The gateway node can temporally cache the fully masked local address so that it does not has to perform a mapping lookup for each packet addressed to D.

4.1.2.5 Fully masked packet delivery

The endpoint S wants to send a fully masked packet to the endpoint D. It is assumed that both endpoints have already registered their fully masked global and local mappings at the GMS and at their $LMSs$ as proposed in Section 4.1.2.3. First, S queries the local mapping system for a fully masked address of D as described in Section 4.1.2.4.

- ◇ If the local mapping system responds with a fully masked local address, the source and destination endpoints reside in the same local domain. In this case, S generates the fully masked packet $< fmLAddr_D \mid fmLAddr_S >$. After that, S performs



(a) Intra-network communication.

(b) Intra-domain communication.

(c) Inter-domain communication.

Figure 4.8: Fully masked packet delivery in BPF-GLI.

$$Test(E(LL_D), T(LL_S))$$

Here, $E(LL_D)$ is D's masked local locator from its fully masked local address. Moreover, $T(LL_S)$ is the trapdoor value of the edge node responsible for the local network to which S is connected. In this way, the source endpoint determines whether the destination endpoint is connected to the same local network, as it does itself.

- ◇ If both endpoints are connected to the same local network (see Figure 4.8a), $Test()$ returns 1. Thus, the source endpoint resolves the masked destination identifier of the packet to a MAC address as explained in Section 4.1.1.1, and it forwards the packet to the destination endpoint.
- ◇ Otherwise (see Figure 4.8b), the packet is first forwarded to the source edge node. For the incoming fully masked packet, a node in the local domain performs

  $$Test(E(LL_D), T(Loc_i))$$

  for each routing table entry i. The packet is then forwarded via the port mapped in the entry for which $Test()$ returns 1. Eventually, the packet arrives at the destination edge node. This node proceeds analogously, and it thus determines that the packet is addressed to the local network for which the node is responsible.

After the resolution of the masked destination identifier, the packet is forwarded to the destination endpoint.

◇ In case of inter-domain communication, the source endpoint gets D's fully masked global address, and it thus generates the fully masked packet $< fmGAddr_D \mid fmLAddr_S >$. In this case, the same packet delivery steps are performed as in GLI-Split (see Figure 4.8c):

1. The packet is forwarded to a gateway responsible for the local domain by using the default route.
2. For the incoming packet from the local domain, the gateway node replaces the masked local locator from the source address of the packet with its own masked global locator, and it thus gets the packet $< fmGAddr_D \mid fmGAddr_S >$. After that, the gateway node performs

$$Test(E(GL_D), T(Loc_i))$$

for each entry i in its masked global routing table. In this way, the gateway node determines the port via which the packet has to be forwarded. Each node in the global domain forwards the packet in the same manner.

3. Eventually, the packet arrives at the destination gateway node. After getting the fully masked local address of the destination endpoint as described in Section 4.1.2.4, the gateway node substitutes $fmGAddr_D$ with $fmLAddr_D$. Thus, the gateway node gets the packet $< fmLAddr_D \mid fmGAddr_S >$. By performing

$$Test(E(LL_D), T(Loc_i))$$

for each entry i in the masked local routing table, the gateway node determines the next hop for the packet. Finally, the packet is forwarded within the destination local domain as explained above.

4.1.3 Alternately blind packet forwarding

BPF-GLI in this mode applies the full blindness in the global domain and the semi-blindness in local domains. Thus, a conventional (unmasked) routing table setup is performed in local domains, while a masked routing table setup is accomplished in the global domain. Hence, a gateway node maintains a masked routing table for the global domain, and an unmasked routing table for the local domain for which the gateway node is responsible.

In case of intra-domain communication, the address fields of a packet from the source endpoint S to the destination endpoint D are constructed as in equation 4.3. If both endpoints

resides in different local domains, the address fields consists of

$$\begin{aligned} &fmGAddr_D \mid smLAddr_S \mid C(ID_S) \text{ in the source local domain,} \\ &fmGAddr_D \mid fmGAddr_S \mid C(ID_S) \text{ in the global domain, and} \\ &smLAddr_D \mid fmGAddr_S \mid C(ID_S) \text{ in the destination local domain.} \end{aligned} \tag{4.9}$$

In this mode, only identifiers are masked in the mapping tables of $LMSs$ as in the semi-blind mode, while both identifiers and locators are encrypted in the mapping table of the GMS as in the fully blind mode. Moreover, the mapping system in a local domain maintains the global locators and PEKS-key pairs of gateways connecting the local domain with the global domain and the fully masked address of the GMS as in the fully blind mode.

In the alternately blind mode, the DHCP server in a local network keeps the cleartext local locator of the edge node responsible for the local network and the semi-masked address of the LMS responsible for the local domain as in the semi-blind mode. In contrast to the semi-blind mode, the server obtains the masked global locators and trapdoors of the gateways and the fully masked address of the GMS from the local domain provider.

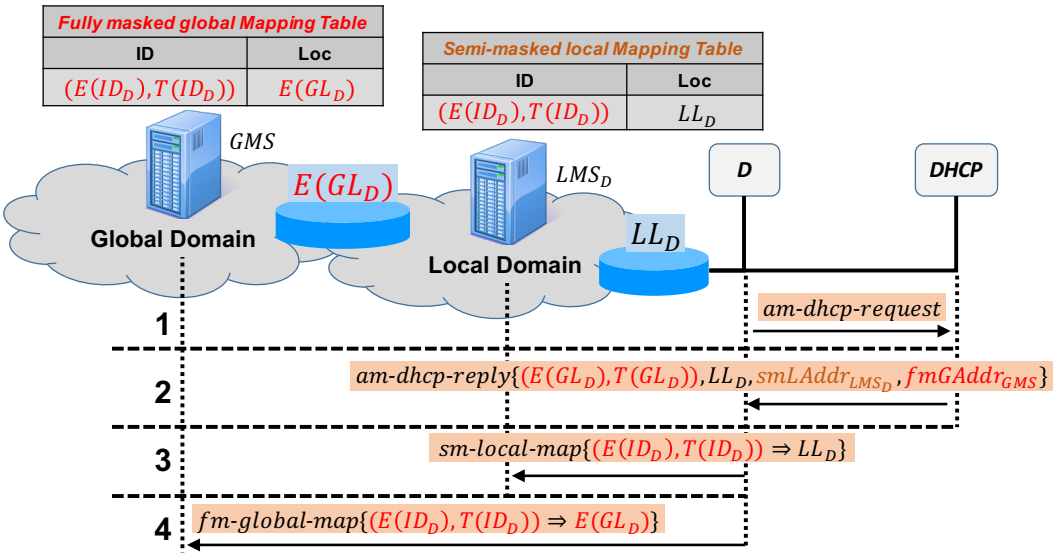


Figure 4.9: Alternately masked mapping registration in BPF-GLI.

4.1.3.1 Alternately masked mapping registration

For the mapping registration in this mode, an endpoint D newly connected to a local network encrypts its identifier with its public key and generates the trapdoor for its identifier with its private key by means of PEKS, and the following steps are performed (see Figure 4.9):

1. The endpoint sends a request to the DHCP server in the local network to which the endpoint is connected.

2. For an incoming request, the DHCP servers responds with the cleartext local locator of the edge node as well as the masked global locator and trapdoor of a gateway responsible for the local domain. Additionally, the alternately masked DHCP reply contains the semi-masked address of the local mapping system LMS_D, and the fully masked address of the GMS:

$$\textit{am-dhcp-reply}\{LL_D, (E(GL_D), T(GL_D)), smLAddr_{LMS_D}, fmGAddr_{GMS}\}.$$

3. The semi-masked local mapping of the endpoint is registered at the LMS_D in the same manner as in the semi-blind mode (see step 3 in Section 4.1.1.3).

4. It is proceeded in the same manner as in the fully blind mode to register the fully masked global mapping of the endpoint at the GMS (see step 4 in Section 4.1.2.3).

The endpoint uses its newly configured semi-masked local address for the semi-masked local mapping registration. Moreover, the source network address of the fully masked global mapping message is the newly configured fully masked global address of the endpoint. Furthermore, the DHCP server can respond with the unmasked addresses of the mapping systems, if the destinations of the registration traffic do not have to be masked. In this case, the endpoint uses them as the destination addresses of the mapping messages.

4.1.3.2 Alternately masked mapping lookup

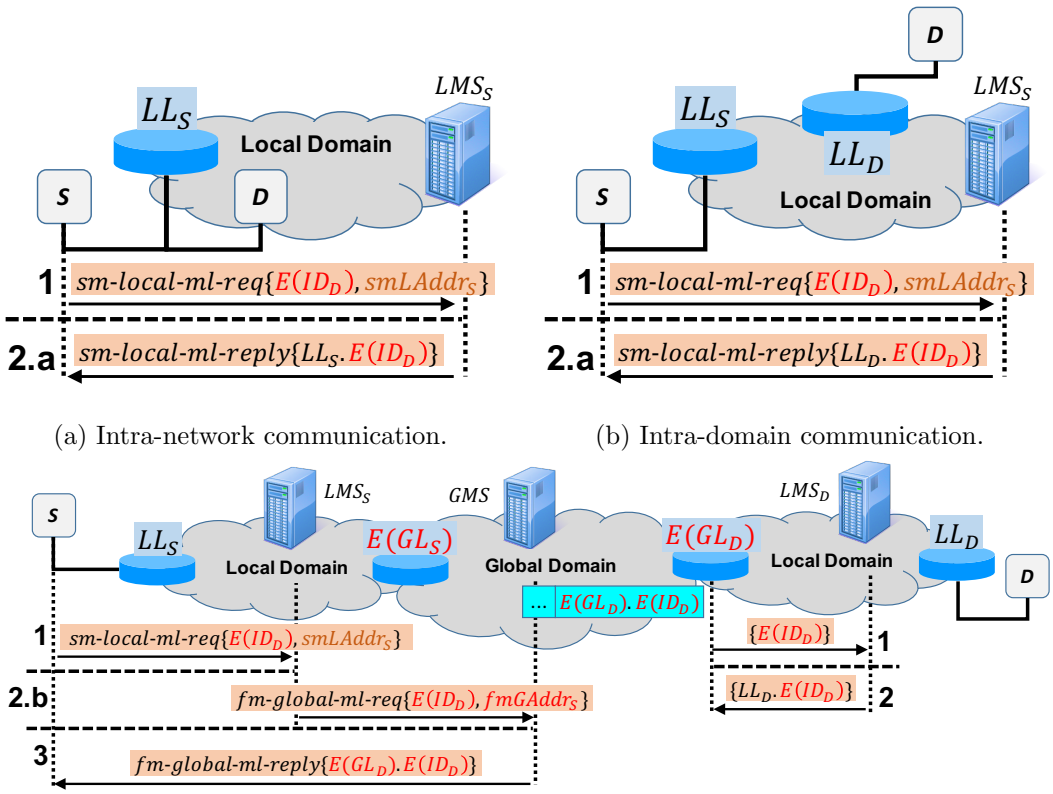
In order to resolve the masked identifier of the destination endpoint D in this mode, the following steps are performed (see Figure 4.10):

1. The source endpoint S sends a semi-masked local mapping lookup request to the local mapping system LMS_S as in the semi-blind mode (see step 1 in Section 4.1.1.4). The request contains D's masked identifier $E(ID_D)$ and S's semi-masked local address $smLAddr_S$.

2. For the incoming request, the LMS_S performs $Test()$ with the masked identifier from the request message and the trapdoor in each mapping table entry.

   (a) If $Test()$ returns 1 for an entry, i.e. both endpoints reside in the same local domain, the mapping system responds with D's semi-masked local address (see Figures 4.10a and 4.10b).

   (b) If $Test()$ returns 0 for all entries, the mode of the lookup request is switched to the fully masked one and the request is forwarded to the GMS (see Figure 4.10c).



(a) Intra-network communication.

(b) Intra-domain communication.

(c) Inter-domain communication.

Figure 4.10: Alternately masked mapping lookup in BPF-GLI.

3. For the incoming fully masked global mapping request, the GMS proceeds in the same manner as in the fully blind mode (see step 3 in Section 4.1.2.4) and responds with D's fully masked global address.

When a gateway receives a fully masked packet which comes from the global domain and contains the masked global locator of the gateway as its destination global locator, the gateway queries the local mapping system LMS_D for the semi-masked local address of the destination endpoint D (see Figure 4.10c). Here, the LMS_D proceeds in the same manner as introduced in Section 4.1.1.4. The gateway can temporally cache the locator for further packets addressed to the endpoint D.

4.1.3.3 Alternately masked packet delivery

For sending a packet from S to D, the source endpoint S first performs a mapping lookup as proposed in Section 4.1.3.2. If the endpoints reside in the same local domain, the source endpoint gets the semi-masked local address of the destination endpoint. Thus, both endpoints pursue a semi-masked intra-domain communication. In this case, S generates the

packet $< smLAddr_D \mid smLAddr_S >$, and the packet is locally forwarded in the semi-masked manner as described in Section 4.1.1.5.

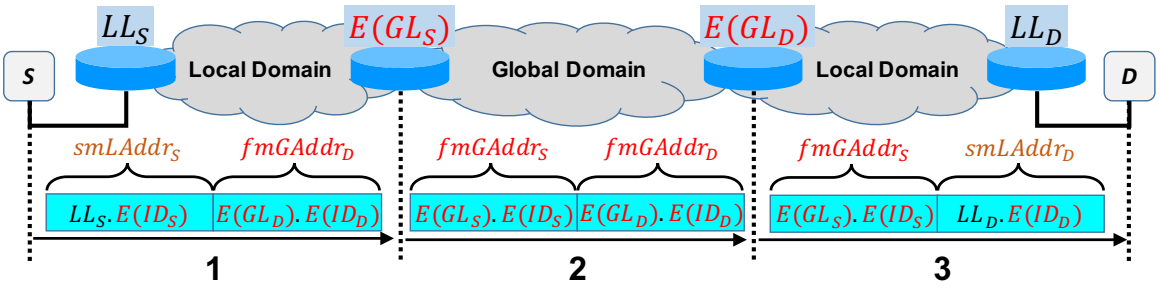


Figure 4.11: Alternately masked packet delivery in BPF-GLI.

In case of inter-domain communication, S gets D's fully masked global address, and it thus generates the packet $< fmGAddr_D \mid smLAddr_S >$. The packet is forwarded in the following steps (see Figure 4.11):

1. The packet is forwarded to the gateway node GL_S by using the default route.
2. When the packet arrives at the gateway node, it replaces the unmasked local locator from the semi-masked source address with its own masked global locator. Thus, the gateway node gets the packet $< fmGAddr_D \mid fmGAddr_S >$. After that, the packet is globally forwarded in the fully masked manner (see step 2 in Section 4.1.2.5).
3. Eventually, the packet arrives at the destination gateway node which first gets D's semi-masked local address from the LMS responsible for the local domain. After that, the gateway node substitutes D's fully masked global address with its semi-masked local address, and gets the packet $< smLAddr_D \mid fmGAddr_S >$. Finally, the packet is forwarded within the local domain in the semi-masked manner (see step 3 in 4.1.1.5).

4.1.4 Analysis

BPF-GLI hierarchically splits the network address of an endpoint into a global and local locator and an identifier, and it replaces the global locator with the local locator at the gateway nodes. Thus, it is well-defined, how many parts an endpoint address has, and which part of the address has to be encrypted with which public key. In this regard, BPF-GLI fulfils the requirement for a hierarchical addressing structure.

Moreover, a source endpoint gets the masked local or global locator of a destination endpoint directly from the local or global mapping system so that the source endpoint needs only

the public key of the destination endpoint in order to encrypt the destination identifier of the packet. Here, the public key of an edge and a gateway node are maintained by the DHCP server and the local mapping system, which are under the control of the local network and domain provider. Thus, we do not need an additional infrastructure to exchange and certify the public keys of network nodes, due to the DHCP server and the mapping system in BPF-GLI.

Additionally, the architecture of BPF-GLI is split into a global domain and multiple local domains so that the core and edge routing is performed separately from each other. Hence, the local routing table entries at a gateway node are aggregated for the global domain by design. However, a further super- or subnetting is not possible within the global domain and local domains in the fully blind mode, while this is the case within all domains in the semi-blind mode, and only within the local domains in the alternately blind mode.

In summary, we can state that BPF-GLI fulfils the architectural requirements for an adequate BPF design with the exception of a fine-grained hierarchical architecture so that performing of the super- and subnetting is not always possible in its entirety. Beside the architectural features of BPF-GLI, this BPF extension supports three modes of blindness providing different network address confidentiality (NAC) levels which can be flexibly adapted to certain use cases. These three blindness modes are discussed in detail below.

4.1.4.1 Semi-blind packet forwarding

BPF-GLI masks the identifier and locator of a network address separately from each other. The semi-blind mode of BPF-GLI encrypts only identifiers end-to-end by means of PEKS, while global and local locators are handled in cleartext. Thus, NAC applies only to the identifier part of a network packet address. We call this semi-NAC property *end-to-end network identifier confidentiality*, or shortly, *network identifier confidentiality (NIC)*.

NIC classifies only the source and destination endpoints as the authorised entities having access to the cleartext identifiers of the network packets transferred between them. However, NIC does not consider the end-to-end confidentiality for the locator part of a network address as its protection target. Thus, the locators of network packets are disclosed to any node on the route as well as to third parties, e.g., potential attackers. Moreover, a network node holds the cleartext locators of other nodes in its routing table entry.

By omitting the third address field in equation 4.3 and 4.4, the identifier of a source endpoint can remain anonymous for the destination endpoint, since the PEKS encryption is not invert-

ible. In this case, the destination endpoint uses the second address field in both equations as the destination address of a reply packet.

While a network node knows the previous and next node of a network packet, the identifiers of the packet are transferred as well as processed in encrypted form by each node on the route of the packet. We discuss NAC's deduced anonymity properties – sender/recipient unlinkability and relationship unlinkability – against local, weak and strong global adversaries in the semi-blind mode of BPF-GLI below.

Local adversary

Since the identifiers of a packet are handled end-to-end in encrypted form, a local adversary, i.e. single network node participating in the transfer of the packet or eavesdropper on that network node can not link the packet to an endpoint as its source/destination endpoint and to a communicating endpoint pair. Hence, sender/recipient unlinkability and relationship unlinkability with regard to endpoints are also the case in the semi-blind mode of BPF-GLI.

However, the adversary knows which local networks communicate with each other in case of intra-domain communication. Thus, the unlinkability properties do not apply to local networks in the semi-blind mode of BPF-GLI. Moreover, the adversary can restrict the set of endpoints possibly communicating with each other to the set of the endpoints connected to local networks which communicate with each other. Nonetheless, the packet still cannot exactly be linked to endpoints. Because of a restricted set of possible communicating endpoints, it is recommended to re-encrypt identifiers at random so that traffic flows between endpoints still remain undistinguishable from each other.

If the communicating endpoints are connected to local networks in different local domains, the local adversary can be located as follows:

- ◇ **In the source local domain:** If the adversary is a network node in the local domain or gateway node responsible for the local domain, in which the source endpoint resides, the adversary can link the packet to its source local network and to its destination local domain, but not to its destination local network, since the adversary gets to see only the destination global locator of the packet. Here, the set of possible communicating endpoints can be restricted to the set of endpoints connected to the source local network and residing in the destination local domain. This set is usually bigger than the set in case of intra-domain communication.

- ◇ **In the destination local domain:** The same applies here as is the case in the source local domain, but in reverse.

- ◇ **In the global domain:** If the adversary is a node in the global domain, it can link

the packet to the communicating local domains as the source and destination local domains of the packet. Thus, the set of possible communicating endpoints consists of the endpoints residing in the local domains which communicates with each other. This set is bigger than the sets above.

Weak global adversary

This kind of a adversary controls a part of network nodes participating in the transfer of a packet. In case of intra-domain communication, the same unlinkability properties as for local adversary apply to weak global adversary. In case of inter-domain communication, if at least one node each in the source and destination local domain of the packet is under the control of the adversary, the adversary knows which local domains as well as networks communicate with each other. If these both nodes are the source and destination edge node of the packet, the adversary knows the exact size of the set of possible communicating endpoints. However, the packet is still not linkable to its source/destination endpoint. Thus, the unlinkability properties with regard to endpoints also apply to weak global adversary.

Strong global adversary

If the adversary controls all nodes taking part in the transfer of a packet, it knows the source and destination local domains and networks of the packet as is the case with weak global adversary and, moreover, the entire route of the packet. Since however the packet identifiers are encrypted end-to-end, the adversary still cannot link the packet to endpoints as its source and destination endpoints.

Mapping system

In the semi-blind global and local mapping system, masked identifiers are mapped to global and local locators in cleartext so that the mapping systems maintain identifiers in encrypted form, and locators in cleartext. Thus, NIC is also the case at the semi-blind global and local mapping systems.

Local mapping systems are under the control of local domain providers. Since identifiers are maintained in local mapping systems in encrypted form, local domain providers do not know which endpoints are connected to the edge nodes in their local domains. Moreover, it is unknown in the global mapping system which endpoints are actually located in which local domains. Furthermore, a mapping lookup request cannot be linked to an endpoint as the requesting endpoint, since the identifier of the requesting endpoint is encrypted in the request message. But due to the cleartext global and local locators of the requesting endpoint in the request message, the global and local mapping systems know from which local domain and network the mapping lookup request originates.

Since the global and local locators are maintained in cleartext by the global and local mapping systems, the current number of endpoints located in a local domain and network is known. Additionally, the global and local locators can eventually disclose information about local domains and networks, e.g., their geographic or organisational positions.

4.1.4.2 Fully blind packet forwarding

In this mode of BPF-GLI, identifiers as well as global and local locators are handled in encrypted form. Thus, NIC also applies here as is the case in the semi-blind mode. Since the locator part of a network packet address is transferred as well as processed end-to-end in encrypted form, NAC applies to the locator part of the network packet address. We call this confidentiality property *end-to-end network locator confidentiality*, or shortly, *network locator confidentiality (NLC)* which classifies only owners of locators of network packets transferred between communicating endpoints as the authorised entities having access to the locators in cleartext.

In case of intra-domain communication, the owners of local locators of packets transferred between two communicating endpoints are the edge nodes to which the endpoints are connected. If the communicating endpoints are connected to local networks in different local domains, packets transferred between them contain global and local locators according to the domain within which the packets are currently forwarded. While the edge nodes responsible for the source and destination local networks possess the local locators in cleartext, the owners of the global locators are the gateway nodes responsible for the source and destination local domains.

While NIC is concerned with the identifier part of a network packet address, NLC handles the locator part of the address. Together NIC and NLC cover the end-to-end transmission and processing confidentiality of the entire network address, and thus offer NAC. Moreover, locators in routing tables of network nodes are handled in encrypted form. Thus, only the owner of a locator has access to the locator in cleartext. The unlinkability properties in the fully blind mode of BPF-GLI are discussed below.

Local adversary

As in the semi-blind mode, sender/recipient unlinkability and relationship unlinkability apply to endpoints against a local adversary. Due to NLC, these unlinkability properties also apply to local domains and networks against the local adversary. Thus, the local adversary cannot link a packet to an endpoint as its source/destination endpoint. Moreover, the packet cannot be linked to a local domain and network as its source/destination local domain and network.

If the local adversary is the source edge or gateway node of a packet, i.e. the adversary is

the owner of the source local or global locator in cleartext, the adversary knows from which local network or domain the packet originates. The same with regard to packet's destination local network or domain applies, if the local adversary is the destination edge or gateway node of the packet.

While a source endpoint itself encrypts the packet identifiers, the source endpoint gets its own locator and the destination locator in encrypted form. If the endpoint always uses the same masked locators for the communication, the network packets transferred between the communicating endpoints can be correlated with each other on the basis of their locators, although the byte values of the masked packet identifiers change by re-encrypting them. In this way, the local adversary can detect a traffic flow between two local domains or networks, while the flow still cannot be linked to a certain local domain or network pair, since the locators of network packets belonging to the flow are encrypted.

In order also to make a traffic flow between local domains and networks indistinguishable, it is recommended that the source and destination endpoints re-register themselves and the source endpoint performs a new mapping lookup at random. Here, the DHCP servers in the source and destination local networks re-encrypt the source and destination local locators and query the local mapping systems in the source and destination local domains for re-encrypted source and destination global locators. Thus, the communicating endpoints get newly encrypted locators.

Weak global adversary

In case of intra-domain communication, if the adversary controls the source and destination edge nodes of a network packet, i.e. the adversary is the owner of the source and destination local locators in cleartext, it knows which local networks communicate with each other. If the communicating endpoints reside in different local domains and, additionally to the edge nodes, the source and destination gateway nodes of the packet are also under the control of the adversary, it also knows which domains communicate with each other. Moreover, the same considerations with regard to traffic flow confidentiality also apply here as are the case with local adversary.

A local domain provider can be also classified as weak global adversary. If the links from local nodes to the edge nodes responsible for the source and destination local networks of a packet are manually configured and it is known which organisations manage the local networks, the local domain provider can link the packet to the local networks in case of intra-domain communication by identifying the route of the packet inside its own domain. Here, the provider monitors all ports of all nodes (in particular, the ports via which the edge nodes can be reached) and compares the masked addresses of all incoming and outgoing packets byte by byte with each other. But the provider still cannot link the packet to endpoints as its source

and destination endpoints, since the identifiers are masked and separated from the masked locators in the packet addresses. In case of intra-domain communication, we can thus state that the unlinkability properties with regard to local networks do not apply against a local domain provider, if the links at the local nodes to the edge nodes are configured manually.

Strong global adversary

This kind of adversary controls all nodes taking part in the transfer of a packet between two communicating endpoints. Since the adversary possesses all locators in cleartext, it can readily link the packet to local domains and networks as its source/destination local domain and network. Moreover, the adversary knows the entire route of the packet. However, the packet is still not linkable to endpoints. Thus, we can state that the unlinkability properties with regard to local domains and networks do not apply against strong global adversary, while these properties are the case for endpoints.

Mapping system

In the fully blind global and local mapping systems, identifiers are handled in encrypted form. Thus, considerations for the semi-blind mapping systems also apply to the fully blind mapping systems with regard to identifiers. In the fully blind global mapping system, global locators are maintained in encrypted form so that information about a local domain is masked from the other local domain providers. Moreover, since local locators are masked in a local mapping system, information about a local network is masked from the provider of the local domain to which the local network is connected. Furthermore, a mapping lookup request cannot be linked to an endpoint as well as to a local network and domain as the requester, since the identifier as well as the local and global locators of the requesting endpoint are encrypted in the request message.

In case of intra-domain communication, a source endpoint gets the fully masked local address of a destination endpoint. Since the local locator of the address is encrypted by means of PEKS which is not invertible, the source endpoint does not know to which local network the destination endpoint is actually connected, and vice versa. In this way, the provider of a local network masks information about its network from the endpoints which aim to communicate with the endpoints connected to the local network. The same applies to inter-domain communication, since the source endpoint gets the fully masked global address of the destination endpoint. Thus, the local domain provider can mask domain information from the endpoints being located in other local domains.

Information about a local domain and network is also masked from the endpoints residing in the local domain and network, since the DHCP server responds with a masked global and local locator generated by the non-invertible PEKS encryption. In summary, NLC and NIC are also the case at the fully blind global and local mapping systems.

4.1.4.3 Alternately blind packet forwarding

In this mode of BPF-GLI, identifiers and global locators of network packets are transferred and processed in encrypted form, while local locators of the network packets are handled in cleartext. Thus, the alternately blind mode provides NIC and supplies NLC only for the global locators of the network packets, while this mode of BPF-GLI is not concerned with the local locators of the packets.

In case of intra-domain communication, only NIC and the same unlinkability properties are provided as in the semi-blind mode. If the communicating endpoints reside in different local domains, NIC and a domain-to-domain NLC apply to the network packets transferred between the endpoints, while only NIC is the case in the local domains. In case of inter-domain communication, the unlinkability properties in this mode are as follows.

Local adversary
The adversary controlling a single network node can be located as follows:

- **In the source local domain:** The adversary can link the packet to its source local network, since the source local locator of the packet is transferred in cleartext. Due to the encrypted destination global locator of the packet, the adversary can however not link the packet to a local domain and network as its destination. Thus, recipient unlinkability with regard to local domains and networks is the case, while sender unlinkability does not apply to local networks. Since recipient unlinkability implies relationship unlinkability, BPF-GLI in this mode supplies relationship unlinkability with regard to local domains and networks against local adversary in the source local domain. Moreover, the packet is not linkable to an endpoint as its source/destination endpoint, since the source and destination identifiers are handled in encrypted form.

- **In the destination local domain:** The same applies here as is the case in the source local domain, but in reverse.

- **In the global domain:** If the adversary controls a node in the global domain, it gets to see only the fully masked source and destination addresses of the packet. Thus, the packet is not linkable to endpoints as well as local domains and networks.

Weak global adversary
If the source and destination gateway nodes of a packet are under the control of the adversary, the adversary possesses the cleartext source and destination global locators of the packet and has access to the source and destination local locators of the packet in cleartext. Thus, the

adversary can link the packet to its source and destination local domains and networks, but still not to an endpoint as its source/destination endpoint.

A local domain provider can readily link a packet to a local network in its domain as the source/destination local network of the packet, since the local locators of the packet are handled in cleartext. Due to encrypted global locators, the provider does not know from which local domain an incoming packet originates and to which local domain an outgoing packet is addressed.

To provide the indistinguishability of traffic flows between local domains, the communicating endpoints can re-register their re-encrypted global locators at the global mapping system and the source endpoint can perform a new global mapping lookup at random as already discussed in 4.1.4.2.

Strong global adversary

If the adversary controls all nodes on the route of a packet, the adversary has access to the source and destination global and local locators of the packet in cleartext. Thus, the adversary can readily link the packet to its source/destination local domain and network. Since however the source and destination identifiers of the packet are encrypted end-to-end, the adversary does not have access to the identifiers in cleartext. Hence, sender/recipient and relationship unlinkabilities with regard to endpoints are the case against strong global adversary.

Mapping system

In the alternately blind mode of BPF-GLI, local mapping systems map encrypted identifiers to cleartext locators, while the global mapping system maintains both identifiers and locators in encrypted form. Thus, only NIC is the case at local mapping systems as in the semi-blind mode, while both NIC and NLC are supplied at the global mapping system as in the fully blind mode.

4.1.4.4 Asymmetric masking

In BPF-GLI, it is also possible to mask the source and destination address of a packet in different modes. For example, if the source address is semi-masked and the destination address is fully masked, the security level of the semi-blindness applies to the source endpoint, while the destination endpoint gets the full blindness. In case of intra- and inter-domain communication, it is disclosed from which local network and domain the packets originate, but it is not known to which local network and domain the packets are addressed. Thus, we do not only support a symmetric NAC with multiple levels, but also an asymmetric NAC for two communicating endpoints.

In this regard, up to nine different masking combinations are possible for a packet between two communicating endpoints residing in different local domains. This means that BPF-GLI provides up to nine various levels of NAC. For example, in a client-server communication, the network address of the server can be semi-masked, if the server (i.e., its global locator) is globally well-known. But if the server (i.e., its local locator) is only locally well-known, the alternate masking of its network address is sufficient for an acceptable NAC level. In case of a private server, performing the full masking for the network address of the server is advisable, especially if the server resides in an unprotected local domain. For the masking mode choice on the client side, beside the sensitivity of the service requested by the client, it is also crucial whether the client resides in a well protected local domain.

The mapping registration mode of an endpoint is crucial for the destination address masking type of a packet to be sent to this endpoint. In this way, the endpoint signals implicitly, in which mode packets have to be forwarded to itself. By means of the mapping lookup, the source endpoint thus learns the requested masking mode for the destination endpoint. Hence, the mapping registration and lookup are also a masking type registration and lookup for an endpoint. The masking mode for a reply packet is determined on the basis of the source address masking type of the packet lastly received. Thus, the source endpoint also signals its masking mode implicitly so that we do not need an additional infrastructure to signal the masking modes requested by endpoints.

4.1.4.5 Network address integrity

Since BPF-GLI does not aim to provide integrity of network packet addresses, the network addresses of a packet can be manipulated unnoticeably. In order also to provide integrity of packet addresses, the network packet header can be expanded by integrity check values for the addresses as proposed in the basic BPF design (see Section 3.2.2).

Due to NIC and its deduced unlinkability properties in the semi-blind mode, an adversary cannot manipulate the addresses of a specific packet on the basis of its source and destination endpoints. But the adversary can target specific local domains and networks, since the packet global and local locators are handled in cleartext. In contrast to that, the adversary can also not specify local domains and networks as attack targets in the fully blind mode, since this mode supplies NIC as well as NLC. The alternately blind mode provides NIC and domain-to-domain NLC so that local domains and endpoints are protected in this regard, while the adversary can still target specific local networks. The same applies to mapping registration and lookup reply messages as well as routing update messages.

A DHCP reply message contains a global and local locator and the addresses of the global and local mapping systems. These values can be also manipulated unnoticeably. Thus, an endpoint can be induced to use manipulated locators, and even to perform mapping registration and lookup at corrupt mapping systems. Therefore, it is recommended also to protect integrity of DHCP reply message values. This can be achieved by integrity check values for these values.

4.1.5 Implementation

In the realisation of BPF-GLI, we have defined two address structures for semi- and full masking. In each of them, a flag defines whether the address contains a global or local locator. Moreover, we have implemented a new network packet header structure (see Figure 4.12) which contains the semi- or fully masked global or local destination and source address fields according to the masking mode choice of the communicating endpoints. Here, two flags define the current masking type of the destination and source address. Additionally, two further flags determines whether the masking type of the destination and source addresses has to alternate for domain crossing. Furthermore, a flag in our header structure states whether the packet has to be forwarded by using the default route. In our implementation, we have omitted the third address field as is the case in the implementation of the basic BPF design.

Multiple new packet structures have been implemented to realise the BPF-GLI functionalities such as mapping registration and lookup, masked identifier resolution, and masked routing. For the link layer discovery, we have extended the packet structure implemented for the basic BPF design by a further flag. This flag defines whether a node sending a link layer discovery packet is a global or local node. In this way, a gateway node can dynamically determine which ports are connected with global or local nodes. Each of these packet types gets a new Ethernet payload type in our implementation. For encryption of locators and identifiers as well as for generation of key pairs and trapdoors we have leveraged the PEKS library [ALPK07].

4.1.5.1 Network side

For realising the BPF-GLI functionalities on the network side, we have expanded the OpenFlow controller NOX by the component *BPF_GLI* which is a subclass of the class *Component* from NOX. The functionalities of a network node are implemented in the C++ class *BPF_GLI_Node*, while the C++ class *BPF_GLI_Gateway* realises the functionalities of a gateway node. An UML diagram of both classes and the associated data structs is given in Figure 4.12. Here, we have omitted the data structures whose UML diagram is already presented in Figure 3.9.

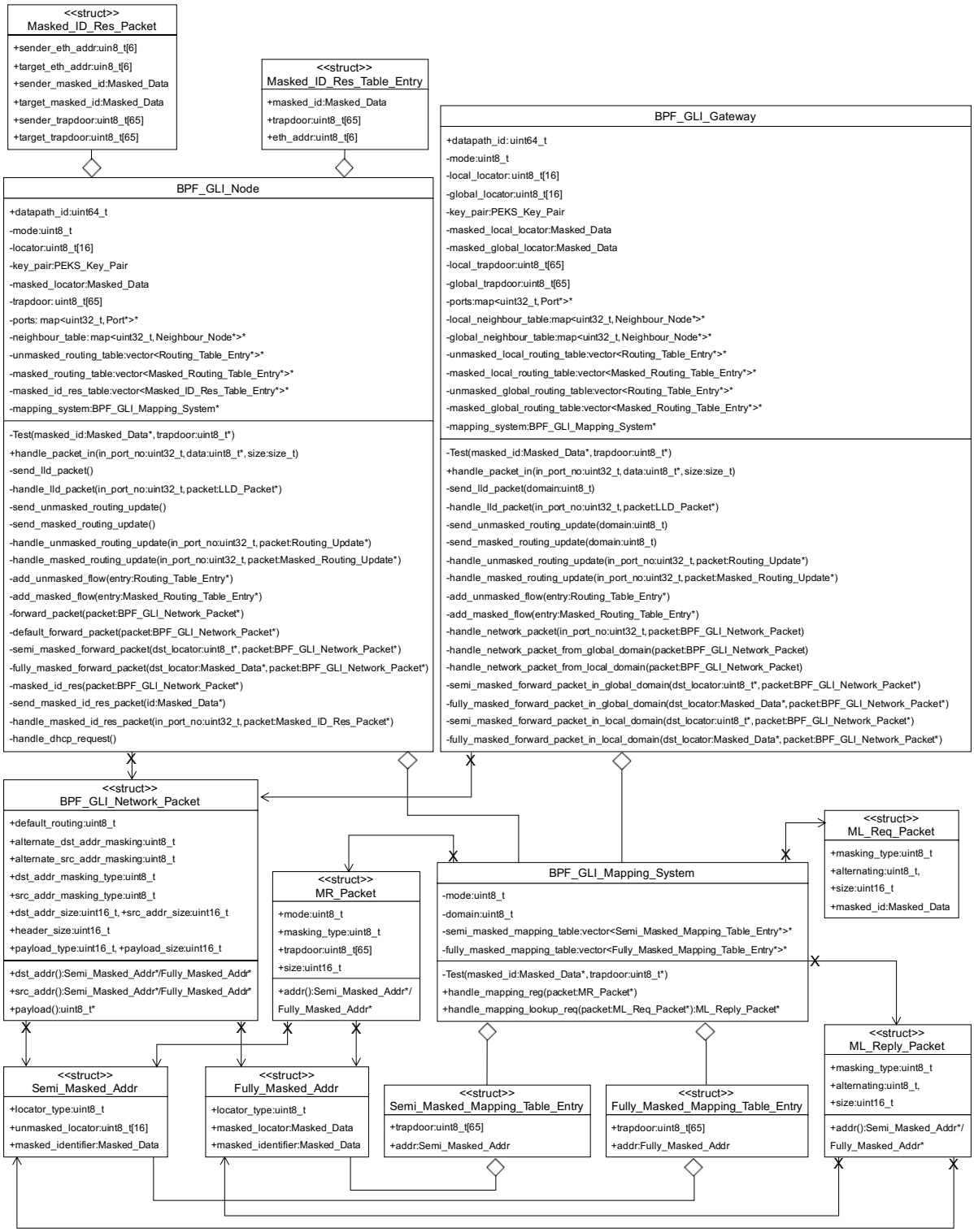


Figure 4.12: UML diagram of *BPF_GLI_Node* and *BPF_GLI_Gateway*.

BPF-GLI splits the entire infrastructure into the global domain and multiple local domains. In principle, the global domain consists of gateway nodes responsible for the existing local domains. Here, each local domain including its local and gateway nodes is under control of a single provider. For each local domain, its own controller component is started in the realisation of BPF-GLI. Thus, the global domain is managed by the controller components for the local domains in a distributed manner. Each controller component maintains a list of objects instantiated from the classes *BPF_GLI_Gateway* and *BPF_GLI_Node*. Moreover, a controller component identifies as well as manages the objects representing gateway and network nodes in the same manner as proposed for the implementation of the basic BPF design in Section 3.3.1.

SDN-like realisation

To realise the network functionalities in BPF-GLI, we have leveraged a SDN-like approach by implementing these functionalities at the controller component. Thus, this relocation of the functionalities, which are usually realised on separate endpoints (i.e., DHCP and mapping servers), to the controller makes it possible that the networks and functionalities are bundled together. In this way, the hosts requiring the functionalities do not interact with separate servers anymore, but rather with the network itself directly. This SDN-like approach improves the performance and makes the network configuration more flexible and dynamic.

Routing: In the implementation of the basic BPF design, the object representing a network node generates a routing update message if necessary. The object then orders the associated OpenFlow datapath to broadcast the message. After a neighbour node receives the message, the node sends it to the controller component which handles the message. Thus, the routing process runs needlessly via the datapaths, although the routing functionalities and tables are implemented at the controller component.

In this regard, we have opted for the SDN-like realisation to implement the routing functionalities in BPF-GLI. Here, after the object representing a network node generates a routing update message, the object calls the function *handle_unmasked_routing_update()* or *handle_masked_routing_update()* on the objects which represent the neighbour nodes of the network node. Thus, the entire routing process runs on the controller, which improves the performance (see Table 4.3). If a neighbour node is managed by another controller, the routing update message is sent via the associated OpenFlow datapath as described above.

Mapping: The mapping functionalities are also implemented at the controller by the C++ class *BPF_GLI_Mapping_System* (see Figure 4.12). Here, the controller component responsible for a local domain creates an object from this class, which represents the local

mapping system in the local domain. Each object representing a gateway or local node in the local domain holds a pointer to this *BPF_GLI_Mapping_System* instance. Moreover, the mapping instance at the controller component has access to the global locators and PEKS-key pairs of gateways of the local domain, which are kept by the objects representing the gateways.

For an Ethernet incoming frame with the type for mapping registration and mapping lookup request, an edge node sends them to the controller. The *BPF_GLI_Node* instance realising the edge node at the controller then calls the functions *handle_mapping_reg()* and *handle_mapping_lookup_req()* on the *BPF_GLI_Mapping_System* object pointer. Thus, the requesting endpoint does not need the network address of the mapping system anymore.

To send a mapping lookup reply back to the requesting endpoint, the controller component orders the OpenFlow datapath from which the request came. Here, the destination Ethernet address of the reply message is the Ethernet address which is cached as the source Ethernet address of the request message. Thus, the requesting endpoint does not need to put its network address into the request message.

If a gateway node receives a packet which comes from the global domain and contains the gateway node's global locator as its destination locator, the gateway node performs a mapping lookup for the destination identifier of the packet. Here, after sending the packet to the controller, the object representing the gateway node at the controller calls the function *handle_mapping_lookup_req()* on the *BPF_GLI_Mapping_System* object pointer.

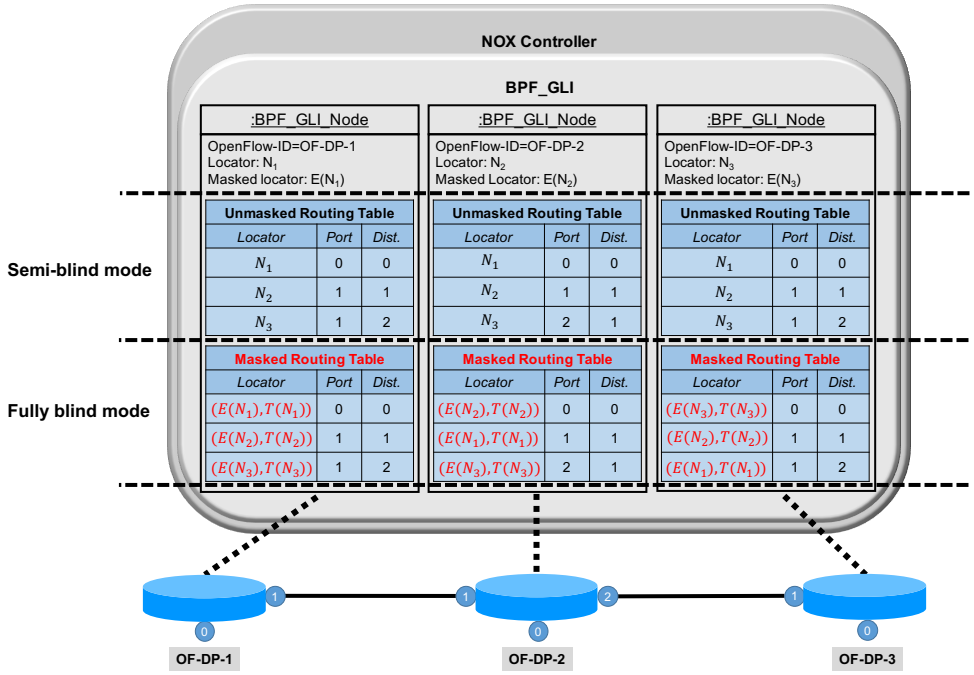
DHCP: The functionalities of a DHCP server are implemented by the function *handle_dhcp_request()* in the class *BPF_GLI_Node*. Thus, a DHCP server does not have to be realised as a separate endpoint. In this way, an edge node receiving a broadcasted DHCP request sends the request to the controller. The object representing the edge node at the controller generates the reply message, and it orders the associated OpenFlow datapath to send the reply message to the requesting endpoint.

Since the endpoint does not need the addresses of the mapping system, the reply message contains only the cleartext locators in the semi-blind mode and the masked locators and trapdoors in the other modes. The unmasked and masked local and global locators as well as the trapdoors of the edge node and the gateways can be obtained from the object representing the edge node and from the mapping system instance at the controller.

Flow-based packet forwarding

In the implementation of the basic BPF design, packets have been handled by the controller component hop-by-hop, since the flow match field types implemented currently in an OpenFlow switch rely on the IP packet structure. However, this approach for packet handling causes a considerable overhead. In order to utilise the benefits of flow-based packet forward-

ing for our implementation, we reinterpret the field type for IPv4-SRC so that the controller component can still define flows for its own purposes. Here, a conventional IPv4 header is added to a BPF-GLI network packet between the Ethernet header and network header. In the IPv4 header, the IPv4-SRC field contains the last four bytes of the destination locator, while the other fields are set to zero. Below, we discuss the setup of flows reflecting routing table entries and the flow-based packet handling and forwarding in case of intra- und inter-domain communication.



Semi-blind mode

Flow Table			
Type	*Length*	*Value*	*Action*
IPv4-SRC	4	last4byte(N_1)	Controller-Output
IPv4-SRC	4	last4byte(N_2)	Output via port 1
IPv4-SRC	4	last4byte(N_3)	Output via port 1

Flow Table			
Type	*Length*	*Value*	*Action*
IPv4-SRC	4	last4byte(N_2)	Controller-Output
IPv4-SRC	4	last4byte(N_1)	Output via port 1
IPv4-SRC	4	last4byte(N_3)	Output via port 2

Flow Table			
Type	*Length*	*Value*	*Action*
IPv4-SRC	4	last4byte(N_3)	Controller-Output
IPv4-SRC	4	last4byte(N_2)	Output via port 1
IPv4-SRC	4	last4byte(N_1)	Output via port 1

Fully blind mode

Masked Flow Table			
Type	*Length*	*Value*	*Action*
IPv4-SRC	4	last4byte($E(N_1)$)	Controller-Output
IPv4-SRC	4	last4byte($E(N_2)$)	Output via port 1
IPv4-SRC	4	last4byte($E(N_3)$)	Output via port 1

Masked Flow Table			
Type	*Length*	*Value*	*Action*
IPv4-SRC	4	last4byte($E(N_2)$)	Controller-Output
IPv4-SRC	4	last4byte($E(N_1)$)	Output via port 1
IPv4-SRC	4	last4byte($E(N_3)$)	Output via port 2

Masked Flow Table			
Type	*Length*	*Value*	*Action*
IPv4-SRC	4	last4byte($E(N_3)$)	Controller-Output
IPv4-SRC	4	last4byte($E(N_2)$)	Output via port 1
IPv4-SRC	4	last4byte($E(N_1)$)	Output via port 1

Figure 4.13: Unmasked and masked flows at nodes in semi- and fully blind mode.

Flow setup: After a cold start of the routing table setup at a node, the controller component orders the associated OpenFlow datapath to define a flow for the node itself (see Figure 4.13). If the node operates in the fully masked mode, the object representing the node

uses the last four bytes of node's masked locator as IPv4-SRC in the match field and sets the action *Controller-Output* so that the matching packets are sent to the controller. In case of the semi-blind mode, the last four bytes of node's locator in cleartext is taken for the match field. Since a gateway node maintains two routing tables, the associated datapath holds two flow tables.

During handling a masked routing update message at a node by means of the function *handle_masked_routing_update()*, the function *add_masked_flow()* is called on the object representing the node for each updated and newly created routing table entry i. This function commands the associated OpenFlow datapath to set up a new flow or to update the flow for the node N_i whose masked routing information is kept by the entry passed as parameter to the function. Here, the function takes the last four bytes of N_i's masked locator $E(N_i)$ as IPv4-SRC in the match field and sets the action *Output* with the port number via which to route to N_i. Defining the unmasked flows is analogous to the masked ones, where the last four bytes of N_i's locator in cleartext are used as IPv4-SRC in the match field. In both modes, a further flow at each local node is defined, which is responsible for default routing. Thus, after the routing has converged, the OpenFlow datapaths hold flow tables reflecting the routing tables of the nodes, which is illustrated in Figure 4.13 for a linear topology consisting of three nodes operating in semi- and fully blind mode.

In the unlikely case that the last four bytes of two masked or unmasked locators are identical, the controller component can take the next to last four bytes. If they are also identical, the controller component takes the third-last four bytes, and so on. However, collisions cannot be avoided completely, since it is tried to compress an entire locator into a field of four bytes. In principle, this test setup serves to demonstrate the possibility of leveraging the flow-based forwarding to a certain extent.

Intra-domain communication: For an incoming BPF-GLI network packet at an edge node, the following steps are performed:

1. The edge node sends the packet to the controller. The object representing the edge node first checks in the function *forward_packet()* (see Figure 4.14) whether the flag for default routing is set. If this is not the case, i.e. the source and destination endpoints reside in the same local domain, the object checks the current masking type of the destination address.

    - ◇ In case of full masking, the function *fully_masked_forward_packet()* is called. This function determines the masked routing table entry i for which *Test()* returns 1.
        - ◇ If $i = 0$, i.e. the communicating endpoints are connected to the same local network, *masked_id_res()* is called to resolve the masked destination identifier to the Ethernet address of the destination endpoint.

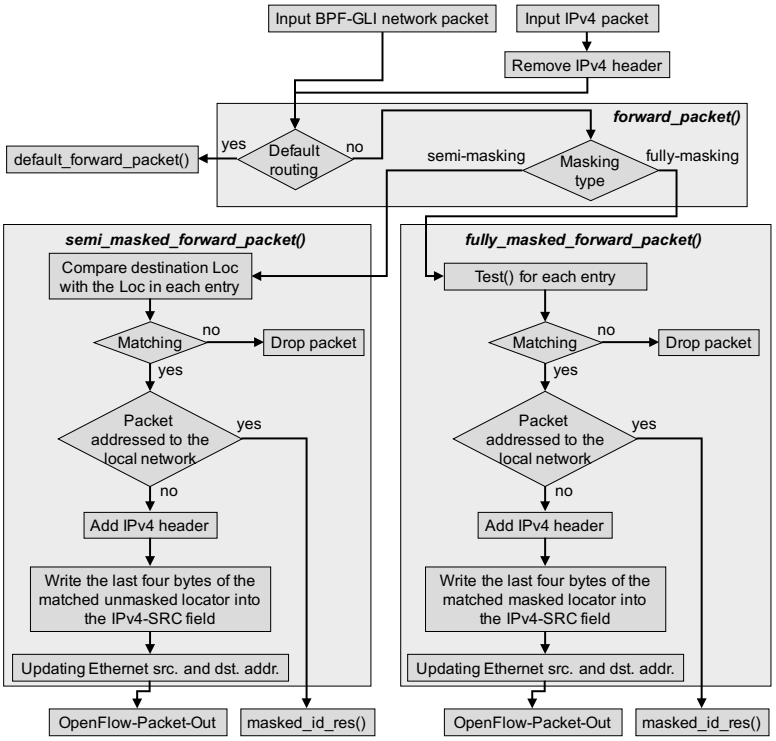


Figure 4.14: Flowchart for packet forwarding at network node.

- ◇ Otherwise, a conventional IPv4 header is added to the packet. Here, the last four bytes of $E(N_i)$ is written into the IPv4-SRC field, while the remaining fields are set to zero.

◇ The object calls the function *semi_masked_forward_packet()* in the semi-blind mode. This function performs the conventional longest prefix matching for the destination locator.

- ◇ If it is determined that the packet is addressed to the local network for which the edge node is responsible, the masked destination identifier is resolved to the Ethernet address of the destination endpoint.
- ◇ Otherwise, the last four bytes of N_i are set analogously.

After handling the packet, the packet becomes an IPv4 packet, and the controller component asks the edge node to forward the updated packet via the port mapped in the entry i. Figure 4.15 illustrates the interaction between an OpenFlow datapath and the controller to handle a semi- and fully masked packet.

2. From then on, the packet is forwarded using the flows defined at the nodes on the route. For an incoming IPv4 packet, an OpenFlow datapath compares the IPv4-SRC

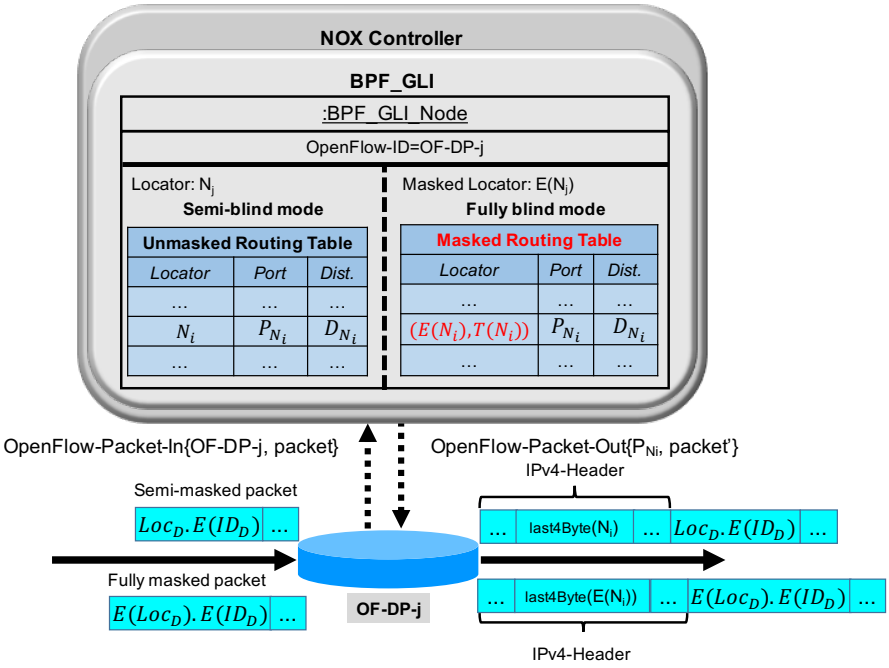
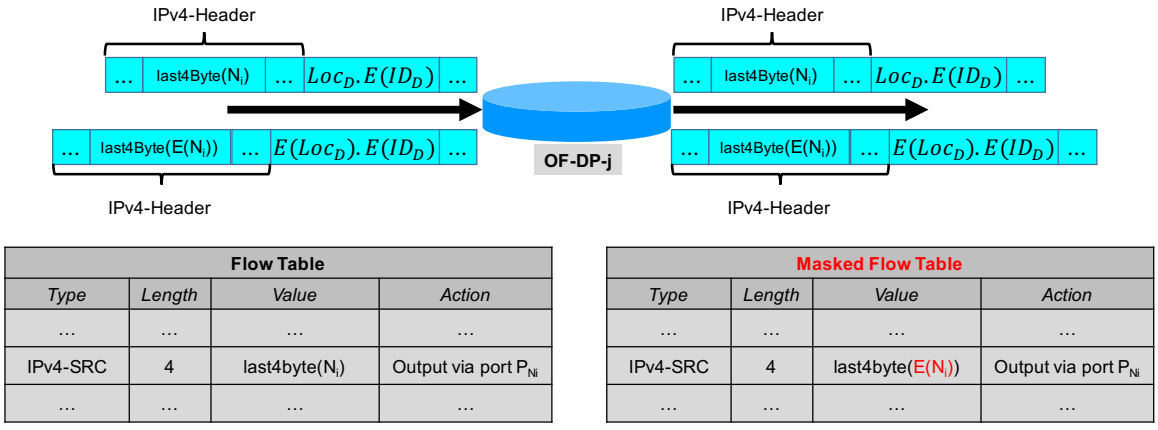


Figure 4.15: Flow-based packet handling in semi- and fully blind modes.



Flow Table			
Type	*Length*	*Value*	*Action*
...	...	...	...
IPv4-SRC	4	last4byte(N_i)	Output via port P_{Ni}
...	...	...	...

Masked Flow Table			
Type	*Length*	*Value*	*Action*
...	...	...	...
IPv4-SRC	4	last4byte($E(N_i)$)	Output via port P_{Ni}
...	...	...	...

Figure 4.16: Flow-based packet forwarding in semi- and fully blind modes.

field of the packet with the match fields of the flows defined in its flow table. If the packet matches a flow, the action defined in the flow is performed so that the packet is forwarded via the port defined in the action. Figure 4.16 shows flow-based forwarding of a semi- and fully masked packet. If the packet does not match any flow, it is sent to the controller to handle the packet as described in step 1.

3. Eventually, the destination edge node receives the packet matching the flow defined

for the edge node itself. Thus, the edge node sends the packet to the controller. After removing the IPv4 header and performing the masked identifier resolution, the controller component orders the destination edge node to forward the updated packet to the resolved MAC address.

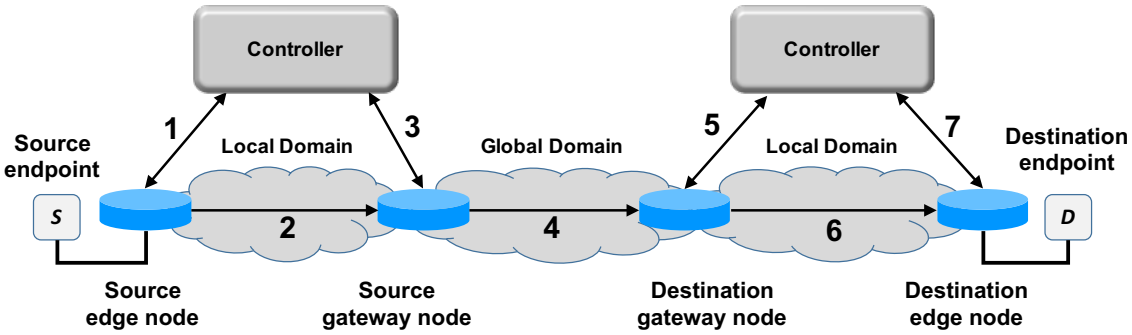


Figure 4.17: Flow-based packet forwarding in inter-domain communication.

Inter-domain communication: If the flag for default routing is set in step 1 above, i.e. the source and destination endpoints reside in different local domains, it is proceeded as follows (see Figure 4.17):

1. The object representing the source edge node adds a conventional IPv4 header to the packet and determines the routing table entry for default routing to a gateway node (source gateway node) responsible for the local domain to which the edge node belongs.

   ◇ In the fully blind mode, the object looks for such an entry in its masked routing table and writes the last for bytes of the masked locator mapped in that entry into the IPv4-SRC field.

   ◇ If the destination address is semi-masked, the objects writes the last four bytes of the cleartext locator mapped in the entry for default routing.

   After that, the controller component orders the associated OpenFlow datapath to forward the packet via the port through which the gateway node can be reached.

2. Until the packet arrives at the source gateway node, it is forwarded using the flows defined at the nodes on the route.

3. At the source gateway node, the packet matches the flow defined for sending the packet to the controller. The object representing the gateway node gets the packet and uses the function *handle_network_packet()* to determine whether the packet comes from the local

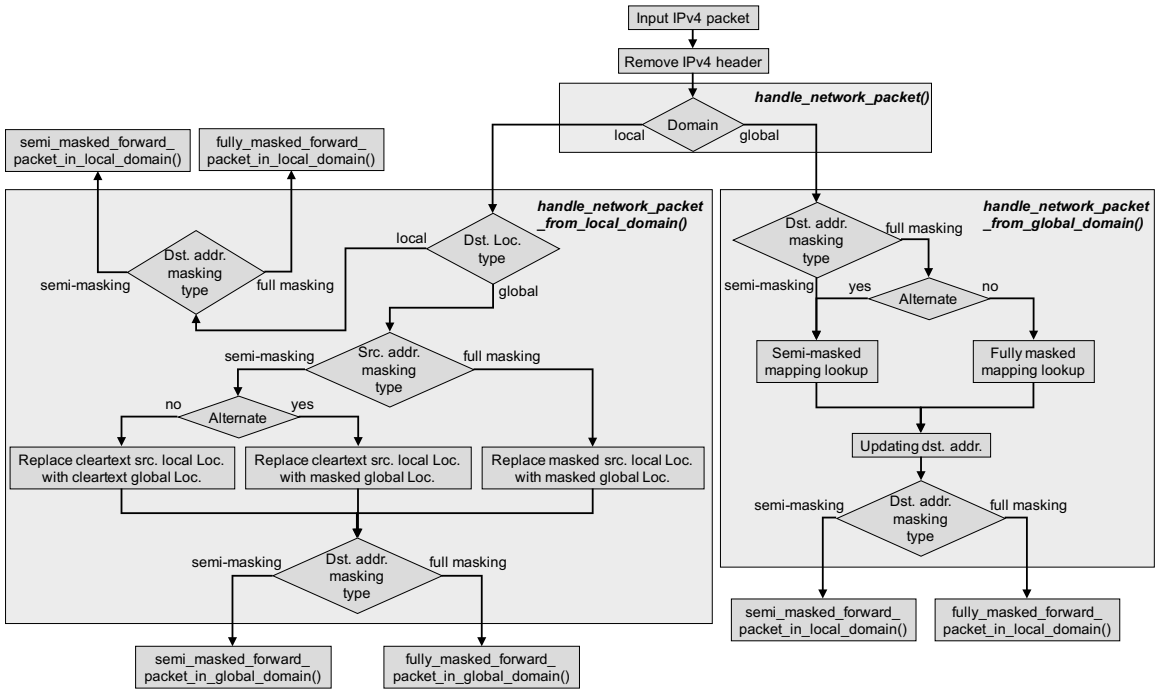


Figure 4.18: Flowchart for packet handling at gateway node.

or global domain (see flowchart in Figure 4.18). Since the first one is the case here, the function *handle_network_packet_from_local_domain()* is called, which checks the type of the destination locator. Since the destination locator of the received packet is a global locator, i.e. the packet has to be forwarded within the global domain, the source locator of the packet has to be replaced with the global locator of the gateway node. Here, the current masking type of the source address has to be checked first.

- ◇ If the source address is fully masked, the masked source local locator is replaced with the masked global locator of the gateway node.
- ◇ In case of semi-masking, the flag for alternating the masking type of the source address is checked.
  - ◇ If this flag is set, the unmasked source local locator is replaced with the masked global locator of the gateway node.
  - ◇ Otherwise, the unmasked global locator of the gateway node is written into the source locator field.

After that, the masking type of the destination address is checked.

- ◇ The function *fully_masked_forward_packet_in_global_domain()* is called in the fully blind mode. This function performs *Test()* for each entry in the masked global

routing table and writes the last four bytes of the masked locator mapped in the entry, for which the method returns 1, into IPv4-SRC field.

- ◇ The function *semi_masked_forward_packet_in_global_domain()* is called in case of semi-masking. This function performs the conventional longest prefix matching for the destination locator and sets the last four bytes of the unmasked locator in the matched entry analogously.

Finally, the flag for default routing is set to zero, and the associated OpenFlow datapath is ordered to forward the updated packet.

4. From then on, the packet is flow-based forwarded within the global domain.

5. Eventually, the packet arrives at the gateway node responsible for the local domain in which the destination endpoint resides. Since the packet matches the flow defined for the gateway node itself, the packet is sent to the controller. Here, after determining that the packet comes from the global domain, the object representing the gateway node calls the function *handle_network_packet_from_global_domain()* to check the current masking type of the destination address (see flowchart in Figure 4.18).

   - ◇ If the destination address is fully masked, the flag for alternating the masking type of the destination address is checked.
     - ◇ In the fully blind mode, i.e. the flag is not set, the object generates a fully masked mapping lookup request for the masked destination identifier. After that, the function *handle_mapping_lookup_req()* is called on the mapping system object pointer to perform a fully masked mapping lookup. After determining the masked local locator of the destination endpoint, the controller component writes this locator into the destination locator field of the packet.
     - ◇ If the masking type of the destination address has to alternate, a semi-masked mapping lookup is performed. After getting the destination endpoint's unmasked local locator, this locator is written into the destination locator field.
   - ◇ If the destination address of the incoming packet is semi-masked, the controller component finds out the unmasked local locator of the destination endpoint. After that, the controller component writes this locator into the destination locator field of the packet in the same way as described above.

   After updating the destination address of the packet, $Test()$ or the conventional longest prefix matching is performed depending on the masking type of the updated destination address. After that, the controller component updates the IPv4-SRC field of the packet accordingly, and it orders the associated OpenFlow datapath to forward the updated packet within the local domain.

6. Until the packet arrives at the destination edge node, it is forwarded flow-based.

7. At the destination edge node, the packet matches the flow defined for the edge node itself. Thus, the edge node sends the packet to the controller which first removes the IPv4-SRC field and subsequently performs the masked identifier resolution. Finally, the object representing the destination edge node orders the associated OpenFlow datapath to forward the packet to the resolved MAC address.

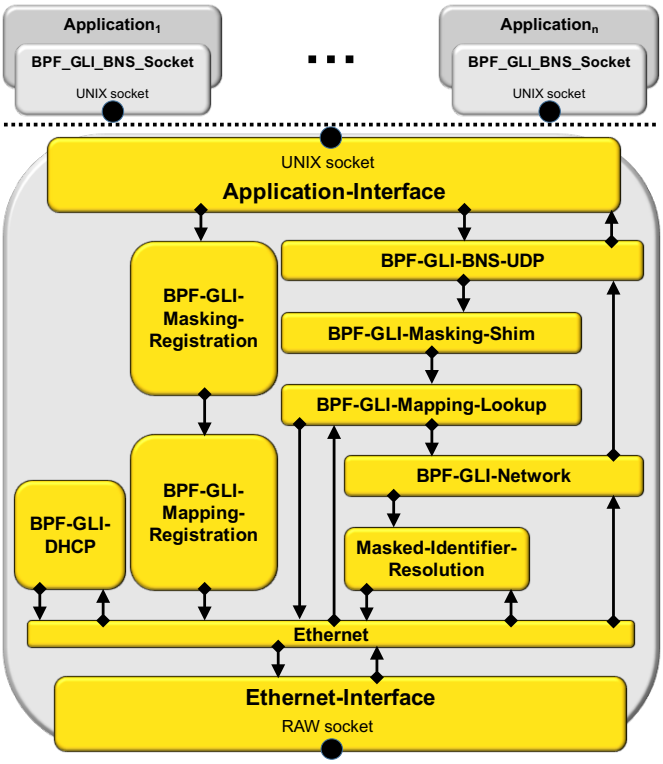


Figure 4.19: BPF-GLI-BNS.

4.1.5.2 Host side

For realisation of the BPF-GLI functionalities on the host side, we have extended the blind network stack (BNS) to *BPF-GLI-BNS* (see Figure 4.19) which is also implemented as a C++ singleton class whose UML diagram is given in Figure 4.20. Here, we have omitted the data structures whose UML diagram is already presented in Figures 3.15 and 4.12. This framework sends and receives Ethernet frames and interacts with applications in the same manner as BNS. For interaction with applications, the interface *Basic_BNS_Socket* has been extended to *BPF_GLI_BNS_Socket* which operates in the same way as the first one.

Figure 4.20: UML diagram of *BPF-GLI-BNS*.

After the initialisation by means of the function *init()*, the BPF-GLI-DHCP module generates a request which is broadcasted in the local network to which the endpoint is connected. For the incoming request, the edge node responsible for the local network responds with its trapdoor value, unmasked and masked local locator, the unmasked and masked global locator and trapdoor of a gateway node responsible for the local domain to which the local network belongs. In this way, the endpoint gets all of the information required to operate in all blindness modes.

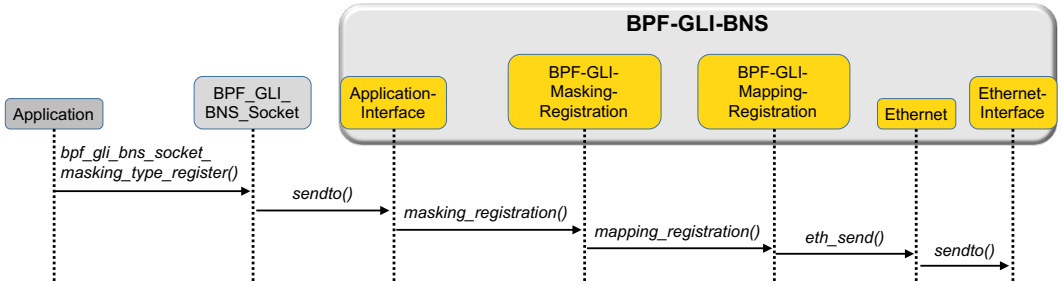


Figure 4.21: Message sequence chart for masking & mapping registration.

To register a masking type requested by an application, it is proceeded as follows (see Figure 4.21):

1. By means of the function *bpf_gli_bns_socket_masking_type_register()*, the application requests *BPF-GLI-BNS* to register the masking type.

2. The masking registration request is passed to BPF-GLI-Masking-Registration module by calling the function *masking_registration()*. This module first checks whether the requested masking type is already registered. If this is not the case, the module generates a new key pair and trapdoor for the endpoint identifier, and it encrypts the identifier with the new public key by means of the PEKS library.

3. Via the function *mapping_registration()*, the masked identifier and trapdoor are passed to the BPF-GLI-Mapping-Registration module which is requested to generate a corresponding mapping registration message depending on the specified masking type.

4. The message is delivered to the Ethernet module for sending it. Thus, the specified mapping is directly registered at the network on the top of the MAC layer.

If an application only possesses the destination identifier, the following processing sequence is performed to send an user payload (see Figure 4.22):

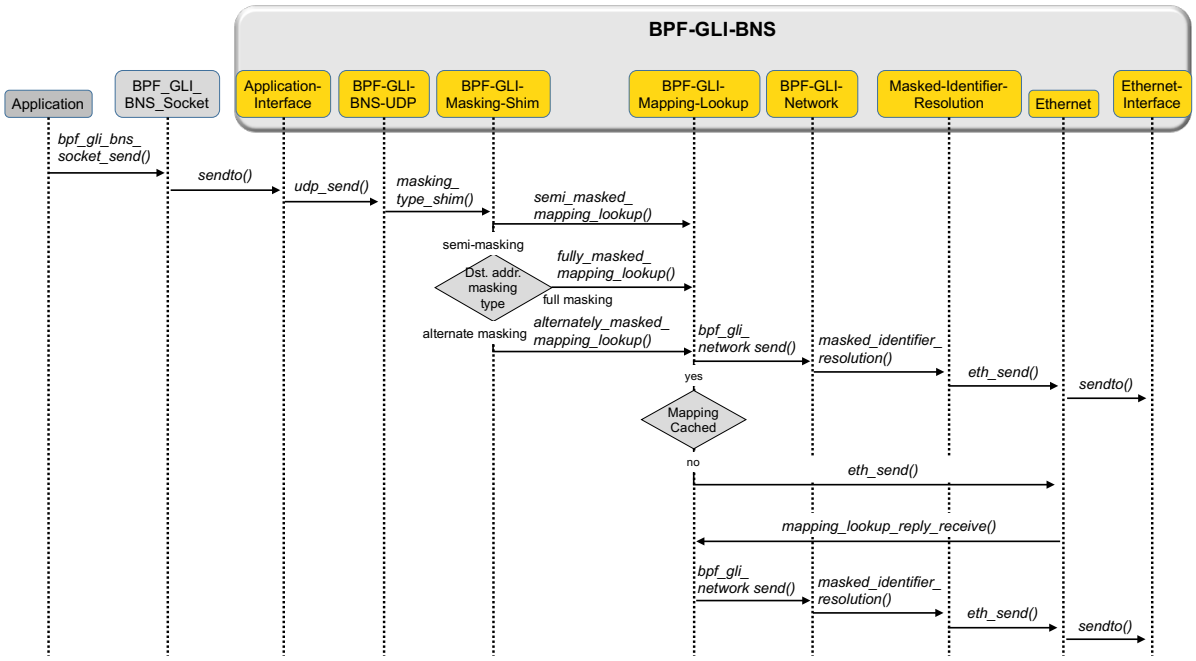


Figure 4.22: Message sequence chart for sending user payload.

1. After masking the destination identifier by means of PEKS, the user application calls the function *bpf_gli_bns_socket_send()*. This function takes the destination BNS-UDP port number, the unmasked and masked destination identifier, the masking type of the source and destination addresses, and the user payload.

2. By calling the function *udp_send()*, the parameters are passed the BPF-GLI-BNS-UDP module. This module generates a BNS-UDP datagram with the received parameters and delivers the datagram to the BPF-GLI-Masking-Shim module via the function *masking_type_shim()*.

3. Depending on the requested masking type for the destination address, the BPF-GLI-Masking-Shim module calls the according function of the BPF-GLI-Mapping-Lookup module.

4. For the incoming parameters, the BPF-GLI-Mapping-Lookup module proceeds as follows:

   ◇ If a mapping for the destination identifier is already cached, the module calls the function *bpf_gli_network_send()* which takes the cached destination address, the masking types of the source and destination addresses, and the BNS-UDP datagram.

- ◇ If no mapping for the destination identifier is found in the mapping cache table, the following steps are performed:
  (a) The BNS-UDP datagram is queued, and the module afterwards generates a mapping lookup request message which is sent by the Ethernet module. For the mapping lookup request, the endpoint thus interacts with the network directly.
  (b) After receiving the mapping lookup reply by the Ethernet module, it delivers the reply to the BPF-GLI-Mapping-Lookup module.
  (c) The BPF-GLI-Mapping-Lookup module first checks whether the masking types in the mapping lookup request and reply match each other. If this is not the case, i.e. the destination endpoint has registered itself in another masking mode, the module adopts the actual masking type of the destination address and caches the mapping.
  (d) The module finds the queued BNS-UDP datagram and passes it including the destination address, and the masking types of the source and destination address to the BPF-GLI-Network module. Before that, the application can be informed about the actual masking mode of the destination endpoint.

5. For the incoming parameters, the BPF-GLI-Network module creates a BPF-GLI network packet header according to the specified masking types of the source and destination addresses and puts it in front of the BNS-UDP datagram. After that, the module checks whether the destination locator is a global or local locator.

   - ◇ If it is a global locator, i.e. the destination endpoint resides in a different local domain, the module sets the flag for default routing and delivers the network datagram to the Masked-Identifier-Resolution module in order to resolve the edge node's Ethernet address.
   - ◇ If the destination locator is a local locator, i.e. the destination endpoint resides in the same local domain as the source endpoint, the module checks whether the source and destination endpoint has the same local locator.
     - ◇ If the source and destination addresses have the same local locator, i.e. both endpoints are connected to the same local network, the network datagram is passed to the Masked-Identifier-Resolution module in order to resolve the destination identifier to an Ethernet address.
     - ◇ Otherwise, the network datagram is handed to the Masked-Identifier-Resolution module for resolving the Ethernet address of the edge node.

6. For the incoming network datagram, the Masked-Identifier-Resolution module proceeds as follows:
   - ◇ If an Ethernet address is already cached for the specified identifier, the network datagram and the cached Ethernet address are given over to the Ethernet module.
   - ◇ Otherwise, the module first queues the network datagram and afterwards generates a masked identifier resolution packet which is sent by the Ethernet module. After the resolution, the module finds out the network datagram and passes it to the Ethernet module which puts an Ethernet header in front of the network datagram and afterwards sends the Ethernet frame.

If the user application possesses the destination identifier as well as the destination locator, e.g., a packet is already received from the endpoint to which the user payload has to be sent, the following steps are performed:

1. The application calls the function *bpf_gli_bns_socket_send_with_dst_loc()*. This function takes the entire destination address, the masking type of the source and destination addresses, and the user payload.

2. Via the function *udp_send_with_dst_loc()*, the parameters are sent to the BPF-GLI-BNS-UDP module which generates a BNS-UDP datagram with the received parameters and delivers the datagram to the BPF-GLI-Network module. From then on, it is proceeded in the same manner as above (steps 5 and 6).

An Ethernet frame carrying a BPF-GLI network datagram is handled as follows:

1. Via the function *eth_receive()*, the frame is passed to the Ethernet module. After removing the Ethernet header, the network datagram is given over to the BPF-GLI-Network module by calling the function *bpf_gli_network_receive()*.

2. The BPF-GLI-Network module first checks the masked destination identifier of the received packet by means of the function *Test()*. After that, the module removes the header and calls the function *udp_receive()* in order to pass the masking types of the source and destination addresses, the source address, and the payload to the BPF-GLI-BNS-UDP module.

3. Finally, the BPF-GLI-BNS-UDP module sends the source BNS-UDP port number, the masking types of the source and destination addresses, the source address, and the user payload to the application listening on the BNS-UDP port whose number is specified in the destination BNS-UDP port number field.

Handling of a DHCP reply packet, a mapping lookup reply packet, and a masked identifier resolution packet is already described above.

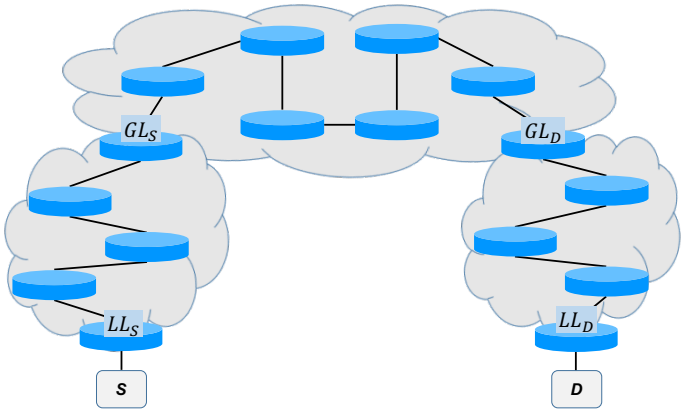


Figure 4.23: Testbed topology for BPF-GLI.

4.1.6 Testbed

The real hardware testbed, in which the implementation of BPF-GLI has been deployed, consists of the same hardware devices, i.e. two PCs and three HP 3800-24G-2SFP+ switches used for the deployment of the basic BPF design (see Section 3.4). On these three HP OpenFlow switches, we have also created 16 OpenFlow instances that are interconnected with each other according to the testbed topology.

The topology for our deployment consists of a global domain and two local domains. Each local domain containing four nodes is connected with the global domain containing six nodes via a gateway node. The nodes in each domain are linearly connected with each other as illustrated in Figure 4.23. Thus, each node in each local domain maintains five routing table entries, while each global node holds eight routing table entries. Moreover, each gateway node has a global and local routing table with eight and five entries. In this topology, we have two endpoints, i.e. S and D that are connected to edge nodes belonging to different local domains. These two endpoints are represented by two instances of *BPF-GLI-BNS* running on the PC with an Intel Core2 Duo 3.33 GHz CPU. Thus, a packet from S to D, or vice versa, takes 16 hops.

For each local domain, its own controller component instance runs on the PC with an Intel Core 2 Extreme 3.06 GHz CPU. In principle, a new instance of the controller component has to be started for each gateway node group which is under the control of another local domain provider. In our deployment, six global nodes are however managed just by a single controller component instance in order to simplify the deployment. In summary, three instances of *BPF_GLI* run on this PC (see Figure 4.24).

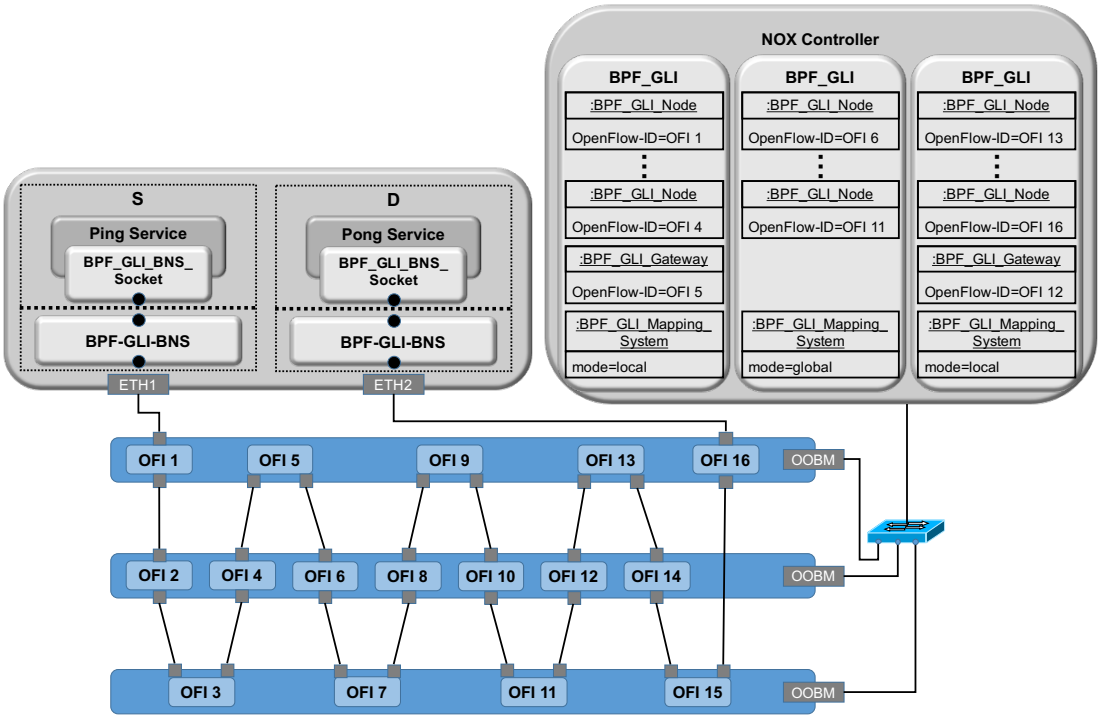


Figure 4.24: OpenFlow testbed for BPF-GLI.

The controller component instance managing the global domain creates an object from the class *BPF_GLI_Mapping_System*. Moreover, each *BPF_GLI_Node* object representing a global node holds a pointer to this *BPF_GLI_Mapping_System* object which represents the global mapping system in this way.

For an incoming global mapping registration, a controller component instance managing a local domain orders the gateway node responsible for that local domain to forward the mapping registration via a global interface. A global node receiving the mapping registration sends it to the controller component which calls the function *handle_mapping_reg()* on the *BPF_GLI_Mapping_System* object.

A global mapping lookup request is handled in a similar manner as a mapping registration. Here, the controller component records the OpenFlow-specific ID of the global node which has received the lookup request, and the number of its port via which the request has come. After resolving the requested identifier to a global locator, the *BPF_GLI_Mapping_System* object generates a mapping lookup reply, and the controller component orders the associated OpenFlow datapath to forward the reply via the recorded port. This port is connected to

a port of the gateway node responsible for the local domain from which the lookup request has come. Thus, this gateway node receives the lookup reply and sends it to the controller component managing that local domain. Finally, the local controller component sends the lookup reply to the requesting endpoint as described in Section 4.1.5.1.

4.1.7 Evaluation

In our implementation, a cleartext identifier/locator consists of 16 bytes corresponding to the size of an IPv6 address, while a masked identifier/locator has 193 bytes, and a trapdoor for an identifier/locator has 65 bytes. Up to 256 bytes, the size of a word in cleartext does not impact the execution times and the output sizes of the operations from the PEKS library used for our implementation. Thus, the benchmarks for the cryptographic operations on the PC with an Intel Core2 Duo 3.33 GHz CPU are the same ones already presented in Table 3.1.

	Size in bytes			***Increase by factor***	
	Unmasked	***Semi-masked***	***Fully masked***	***Semi-masked***	***Fully masked***
Address	33	210	387	6.36	11.72
Network header	81	435	789	5.37	9.74
Mapping reg. packet	37	279	456	7.54	12.32
Mapping lookup req. packet	20	197	197	9.85	9.85
Mapping lookup rep. packet	37	279	456	7.54	12.32
Mapping table entry	33	275	452	8.33	13.69
Mapping lookup cache entry	57	494	671	8.66	11.77

Table 4.1: Sizes of unmasked, semi-, and fully masked structures.

Table 4.1 shows the sizes of unmasked, semi-, and fully masked structures implemented for BPF-GLI. The structure sizes for unmasked and masked routing as well as identifier resolution are presented in Table 4.2. Moreover, both tables present the size increasing factors in comparison with the unmasked structures. Thus, BPF-GLI introduces a size overhead increase by a factor 7.66 and 11.63 in average for semi- and full masking.

In the testbed topology described in Section 4.1.6, the convergence times for the unmasked and masked routing table setups which are implemented conventional and SDN-like, are given

	Size in bytes		***Increase***
	Unmasked	***Masked***	***by factor***
Routing update entry	18	260	14.44
Identifier resolution packet	46	530	11.52
Routing table entry	28	272	9.71
Identifier resolution table entry	22	264	12

Table 4.2: Structure sizes for unmasked and masked routing and identifier resolution.

in Table 4.3. Here, we can see that the SDN-like realisation can achieve a significant performance increase for the routing table setup. Moreover, this table demonstrates the benefit of the routing separation in terms of the convergence time (compare with Table 3.2). The same table also shows the execution times for resolving an unmasked and a masked identifier to a MAC address in case of an empty neighbour cache. Furthermore, this table shows the time increasing factors for the masked functionalities in comparison with the unmasked ones.

		Time in milliseconds		***Increase***
		Unmasked	***Masked***	***by factor***
Conventional routing table setup	***global***	53.60	3620.31	67.54
	local	35.16	1501.15	42.69
SDN-like routing table setup	***global***	34.43	3195.44	92.80
	local	14.45	1041.64	72.08
Identifier resolution		1.30	4.78	3.67

Table 4.3: Execution times for routing and identifier resolution.

We have evaluated the performance of the controller component *BPF_GLI* and the network stack *BPF-GLI-BNS* in a ping-pong scenario in the testbed topology. By means of the conventional as well as the semi-, fully, and alternately blind packet forwarding, the endpoint S pings the endpoint D ten times for each case, while D responds by sending a pong packet back to S for every ping packet received.

Figures 4.25 and 4.26 show the individual RTTs for the unmasked, semi-, fully, and alternately masked ping and pong packets which are handled hop-by-hop and flow-based. Because only the identifiers are masked, the semi-blind mode performs better than the fully blind one. Moreover, the same applies for the alternate masking, since the packet addresses are fully masked only in the global domain.

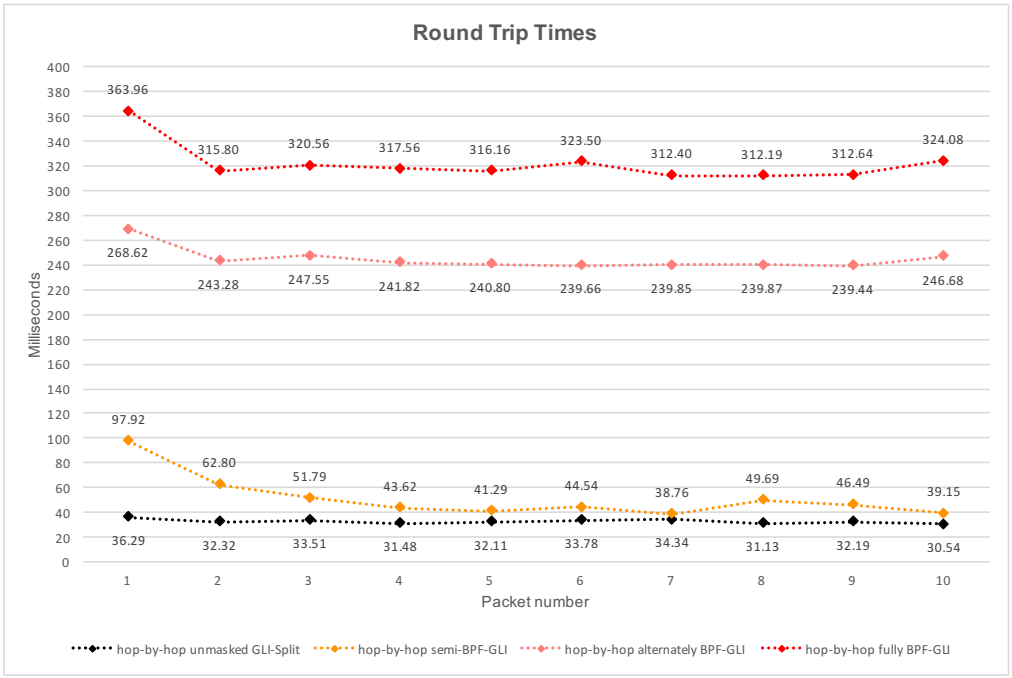


Figure 4.25: Round trip times for packets handled hop-by-hop.

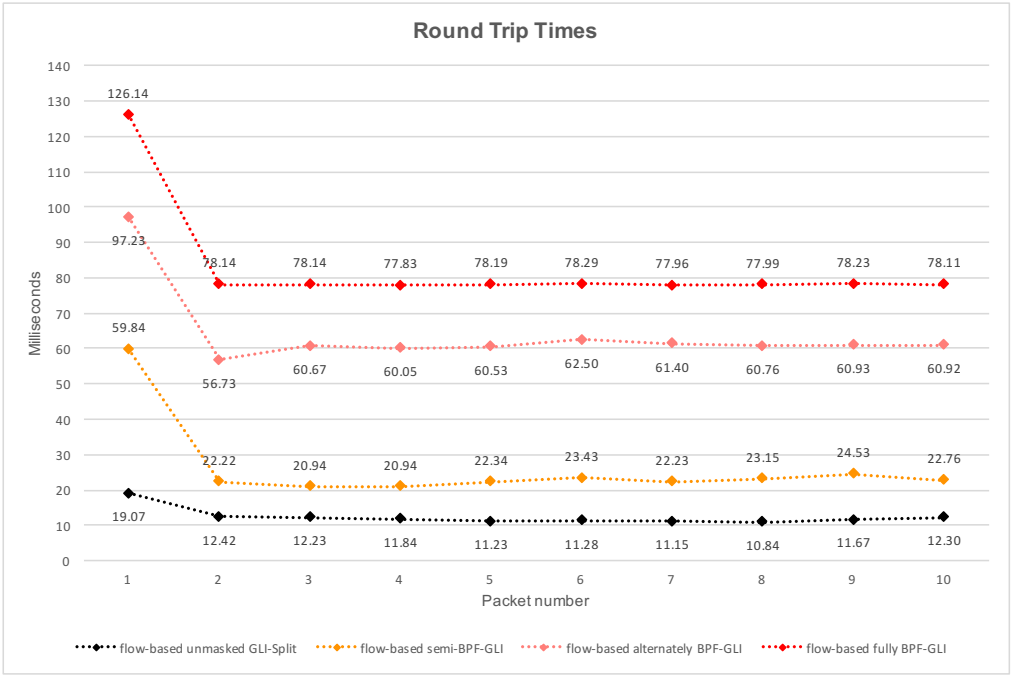


Figure 4.26: Round trip times for packets handled flow-based.

In each case, the round trip for the first packet takes more time than for the remaining packets, since the source endpoint S and the destination gateway node GL_D have first to resolve D's global and local locator. These RTT differences thus represent the execution times for mapping lookups in the semi-, alternately, and fully blind modes. S and GL_D cache the locators so that they do not need to perform a mapping lookup for the remaining packets.

This basic experiment demonstrates the benefits of the flow-based packet forwarding compared to hop-by-hop packet handling by the controller. Here, we especially want to point out that the flow-based alternately BPF-GLI is just about two times slower than hop-by-hop unmasked GLI-Split. Thus, flow-based alternately BPF-GLI, which provides NIC and domain-to-domain NLC, seems to be promising to be realised in practice.

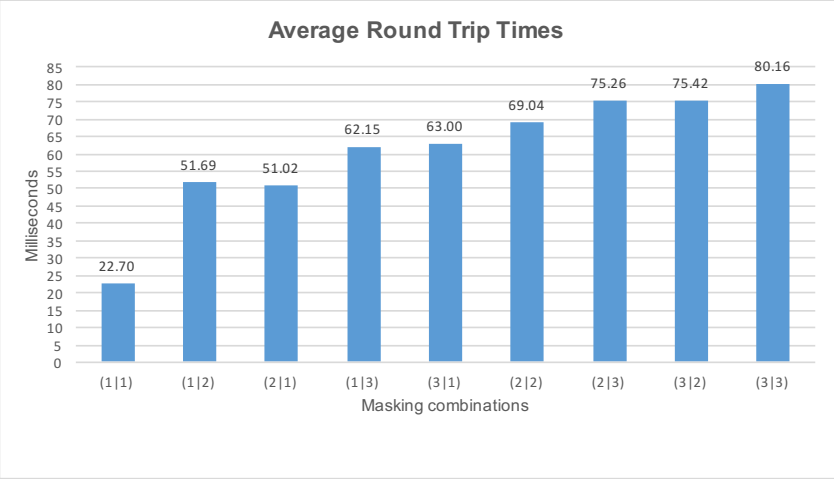


Figure 4.27: Average round trip times for masking combinations.

As already discussed in Section 4.1.4.4, the communicating endpoints do not necessarily have to mask their addresses symmetrically. Figure 4.27 shows the average RTTs of ten packets that are transferred flow-based between the endpoints S and D in the testbed topology. Here, we have applied nine possible masking combinations to each ten packets. In this figure, 1, 2, and 3 denote the semi-, alternate, and full masking of the destination or source address. For example, (1 | 3) thus denotes that the destination address is semi-masked, while the full masking is applied for the source address. The dashed line in Figure 4.27 represents the individual overheads introduced by the respective masking combinations. Here, similar average RTTs apply to the reversed masking combinations such as (1 | 2) and (2 | 1). In summary, we can state that the stronger the NAC level applied to the packets, the higher is the average RTT for these packets.

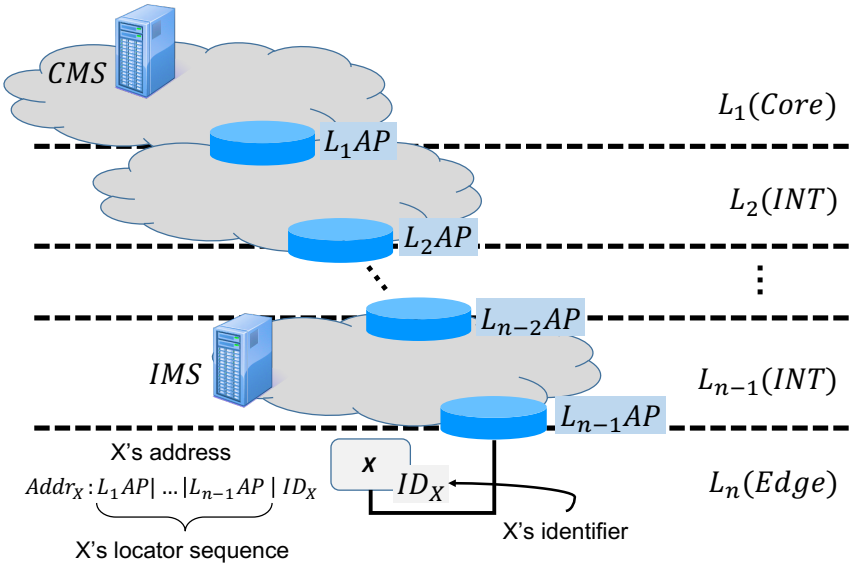


Figure 4.28: HAIR's n-level-based hierarchical scheme.

4.2 BPF in a hierarchical Architecture

The *Hierarchical Architecture for Internet Routing (HAIR)* [FCM+09] is a FNA approach which relies on the Loc/ID Split principle and has been sketched in Section 2.2.2. This approach defines a n-level-based hierarchical scheme which reflects the current Internet structure (see Figure 4.28). There, local networks placed at level n are called *Edges*. At levels $n-1$ to 2 we have the so called *Intermediate (INT)* networks which consist only of routers and ensure the reachability between the attached Edges and INTs of the next higher level. We can conceive the INTs as access providers or enterprise networks. The top level of the hierarchy is the so called *Core* providing routing reachability between the INTs at level 2. Each router in the Core belongs to one administrative domain and is under the control of a single ISP, just like in the today's Internet backbone. The organisation of hierarchy levels is not restricted to be globally symmetrical. Domain providers can independently organise their own domains in various hierarchy levels.

Levels 1 to n are connected via so called *Level Attachment Points (LAPs)* acting as gateway nodes. A routing domain at level k is connected to a routing domain at level $k+1$ via L_kAP. The address of an endpoint consists of its identifier and actual locator sequence. This sequence defines the LAPs to be traversed for forwarding a packet from the Core to the destination Edge, which is similar to loose source routing. At the Edge, the destination identifier is resolved to the MAC address of the destination endpoint, and the packet is forwarded to the destination endpoint.

In order to bind the actual locator sequence to the identifier of an endpoint, HAIR defines a hierarchical mapping system consisting of a *Core Mapping Service (CMS)* and multiple *INT Mapping Services (IMSs)*. While an IMS holds the actual mappings for the endpoints belonging to that INT, the CMS keeps a pointer to the IMS maintaining the current mapping for each identifier. Here, an IMS is under the control of a $L_{n-1}(INT)$ provider, and the CMS can be maintained by the $L_2(INT)$ providers in a distributed manner. A new endpoint at an Edge has to register the mapping of its identifier to its locator sequence at the mapping system. Before sending a packet, a source endpoint has to resolve the identifier of the destination endpoint to its actual locator sequence. For that, the source endpoint queries the mapping system.

In this section, we adopt the addressing structure and architectural features of HAIR and extend the basic BPF design to a new design called *Blind Packet Forwarding in Hierarchical Architecture for Internet Routing (BPF-HAIR)*. This BPF extension defines two modes, namely semi- and fully blind modes which perform the same processes as defined in HAIR for packet delivery. After presenting the construction, Section 4.2.3 analyses BPF-HAIR. The implementation and evaluation of BPF-HAIR are discussed in Sections 4.2.4 and 4.2.6.

For the construction, we assume that all network nodes, communicating endpoints, and mapping systems have already generated a key pair using PEKS. Moreover, it assumed that the public keys of the communicating endpoints are already exchanged and bound to their owners. Furthermore, we make the assumption that the communicating endpoints have already found the identifiers of each other, e.g., via DNS.

4.2.1 Semi-blind packet forwarding

This mode of BPF-HAIR only masks identifiers. Thus, the semi-masked address of the endpoint X with the identifier ID_X is

$$smAddr_X : L_1AP_X|...|L_{n-1}AP_X \mid E(ID_X),\ where \tag{4.10}$$

- $E(ID_X) = PEKS(X_{pub}, ID_X)$ is X's masked identifier generated with X's own public key X_{pub}.
- $Loc_X = L_1AP_X|...|L_{n-1}AP_X$ is X's actual locator sequence in cleartext.

The semi-masked address fields in a packet from the source endpoint S to the destination endpoint D consist of their semi-masked addresses generated according to equation 4.10 and

the ciphertext generated by conventionally encrypting (e.g., with RSA) ID_S with the corresponding public key of D:

$$smAddr_D \mid smAddr_S \mid C(ID_S). \tag{4.11}$$

Here, the last field serves the same purpose as described in BPF-GLI (see Section 4.1.1), namely getting the source identifier in cleartext. For semi-masking a packet, the source endpoint encrypts the source identifier with its own public key and the destination identifier with the public key of the destination endpoint. The semi-masked source address of a packet is used as the destination address of a corresponding destination unreachable message, if the packet cannot be forwarded to the destination endpoint. The destination endpoint uses the second address field as the destination address of a reply packet. The communicating endpoints can cache the masked identifiers so that they do not have to encrypt the identifiers for each packet.

Due to handling locators in cleartext, a conventional (unmasked) routing algorithm such as Distance Vector Routing algorithm suffices to set up routing tables. But here the unmasked routing is performed domain-wise so that a network node in a routing domain at level k maintains routing entries only for the network nodes in the same domain at level k, $1 \leq k \leq n-1$. Moreover, a LAP connecting the routing domains at level k and $k+1$ holds a routing table for each of the domains. For resolving a masked identifier to a MAC address, we adopt the masked identifier resolution approach from BPF-GLI (see Section 4.1.1.1). The redesigning of mapping functionalities and packet delivery are presented below.

4.2.1.1 Semi-blind mapping system

The mapping system architecture of HAIR is adopted by the semi-blind mode of BPF-HAIR. In this mode, only identifiers are encrypted in the mapping tables of the CMS and $IMSs$, while locators are handled in cleartext. In the IMS responsible for the INT in which the endpoint X resides, a *semi-masked mapping table entry (sm-MTE)* for the endpoint X consists of its masked identifier, trapdoor, and locator sequence:

$$\textit{sm-MTE}_X : [(E(ID_X), T(ID_X)), L_1AP_X|...|L_{n-1}AP_X].$$

In HAIR, an enhanced DHCP server at an Edge holds the locators of LAPs via which the Edge can be reached from the Core. In contrast to that, the DHCP server keeps only the cleartext locator of the edge node in BPF-HAIR. Moreover, the IMS in the INT to which the Edge is attached maintains the cleartext locators of remaining LAPs (i.e., LAPs at levels 1 to $n-2$) via which the IMS can be reached from the Core. This entry can be manually configured or dynamically set up by running an enhanced traceroute, e.g., for the CMS. Furthermore, the DHCP server keeps IMS's semi-masked address, and the IMS holds the

semi-masked address of the CMS. If the destinations of the mapping registration and lookup request messages do not have to be masked, only the cleartext addresses are maintained.

The CMS maintains mapping pointers to $IMSs$ holding the actual mappings for endpoints. In the CMS, a *semi-masked mapping pointer entry (sm-MPE)* for the endpoint X consists of its masked identifier, trapdoor, and the semi-masked address of the IMS keeping the mapping for the endpoint X:

$$sm\text{-}MPE_X : [(E(ID_X), T(ID_X)), smAddr_{IMS_X}].$$

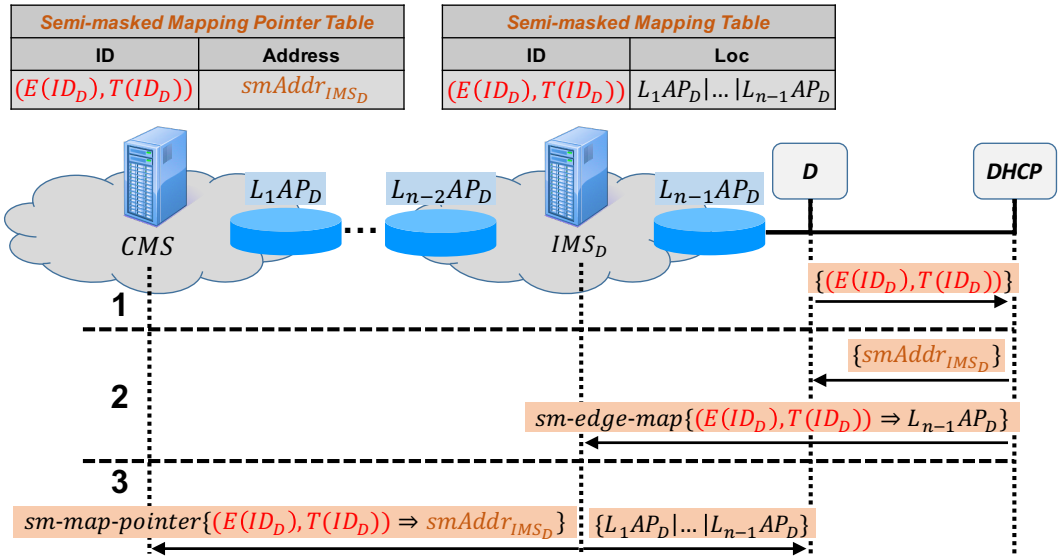


Figure 4.29: Semi-masked mapping registration in BPF-HAIR.

4.2.1.2 Semi-masked mapping registration

The identifier of an endpoint newly connected to an Edge has to be reported to the IMS responsible for that Edge and to the CMS so that the identifier is bound to a locator sequence. HAIR does not specify how this process has to be performed. In order to register the mapping of the masked identifier of the new endpoint D to its cleartext locator sequence, BPF-HAIR proceeds as follows (see Figure 4.29):

1. The endpoint D encrypts its identifier with its public key and generates the trapdoor value for its identifier with its private key by means of PEKS. After that, D sends its masked identifier and trapdoor to the DHCP server.

2. The DHCP server first responds with the semi-masked address of the IMS responsible for the INT to which the Edge is attached (IMS_D). Here, it is also possible that the

endpoint gets the cleartext address of the IMS_D. Next, the DHCP server maps the edge-locator $L_{n-1}AP_D$ to D's masked identifier and trapdoor and sends the semi-masked edge-mapping to the IMS_D:

$$\textit{sm-edge-map}\{(E(ID_D), T(ID_D)) \Rightarrow L_{n-1}AP_D\}.$$

For the incoming edge-mapping, the IMS_D performs $Test(E(ID_i), T(ID_D))$ for each entry i.

- ◇ If $Test()$ returns 1 for an entry, i.e. a mapping for the endpoint D already exists, the last locator in the sequence in the entry is replaced with the edge-locator sent from the DHCP server. Moreover, the masked identifier in the entry is replaced with the masked identifier from the edge-mapping in order to update the byte value of the ciphertext for the identifier.
- ◇ If $Test()$ returns 0 for all entries, i.e. no entry for the endpoint D exists so far, a new entry is created with the part of the locator sequence held already at the IMS_D and the edge-locator sent from the DHCP server.

3. If a new entry for the endpoint is created at the IMS_D, it sends the entire locator sequence back to D's semi-masked edge-address $L_{n-1}AP_D \mid E(ID_D)$. Moreover, the IMS_D sends the semi-masked mapping pointer consisting of the masked identifier and trapdoor of the endpoint as well as IMS_D's semi-masked address to the CMS:

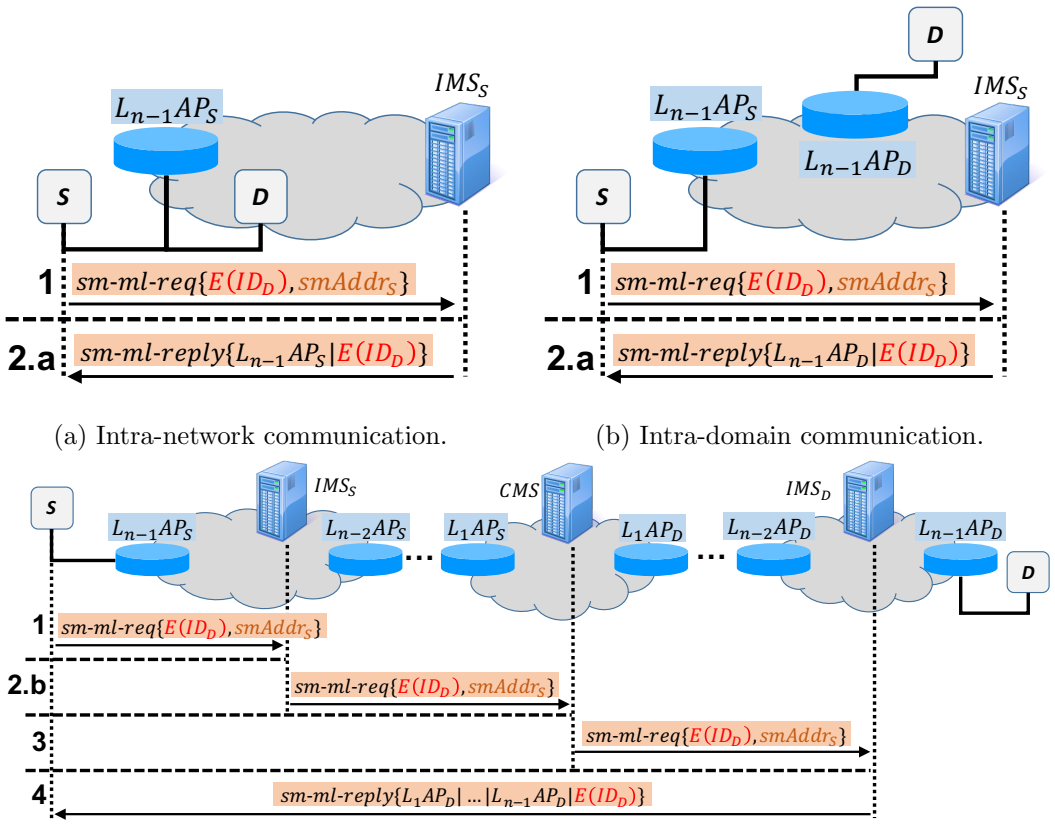
$$\textit{sm-map-pointer}\{(E(ID_D), T(ID_D)) \Rightarrow smAddr_{IMS_D}\}.$$

When D's semi-masked mapping pointer arrives at the CMS, it performs $Test()$ with D's trapdoor and the masked identifier in each entry as parameters. If $Test()$ returns 1 for an entry, the CMS updates the masked identifier and the semi-masked address in the entry with the values from the mapping pointer message. Otherwise, a new entry is created for the endpoint.

Here, we want to point out that the DHCP server, IMS_D and CMS can communicate with each other by using their cleartext or semi-masked addresses. Moreover, the semi-masked mapping pointer of the endpoint D can contain the cleartext address of the IMS_D.

4.2.1.3 Semi-masked mapping lookup

For sending a packet to the destination endpoint D, the source endpoint S needs D's actual locator sequence. As in the mapping registration, HAIR does not specify how to proceed for that. In order to resolve D's masked identifier to its actual locator sequence in cleartext, the following steps are performed in BPF-HAIR (see Figure 4.30):



(a) Intra-network communication.

(b) Intra-domain communication.

(c) Inter-domain communication.

Figure 4.30: Semi-masked mapping lookup in BPF-HAIR.

1. After masking D's identifier by means of PEKS, the source endpoint generates a semi-masked mapping lookup request. This message contains D's masked identifier $E(ID_D)$ and S's semi-masked address as the source address of the request message. After that, the endpoint S sends the request message to the IMS at which it is registered (IMS_S):

$$sm\text{-}ml\text{-}req\{E(ID_D), smAddr_S\}.$$

2. For the incoming request, the IMS_S performs $Test(E(ID_D), T(ID_i))$ for each mapping table entry i.

   (a) If $Test()$ returns 1 for an entry, i.e. the endpoints S and D are located in the same INT, the IMS_S responds with D's semi-masked edge-address (see Figures 4.30a and 4.30b). Here, the locator sequence in this edge-address consists only of D's edge locator.

   (b) Otherwise, the IMS_S forwards the request to the CMS (see Figure 4.30c).

3. For the incoming request, the CMS performs $Test()$ with the masked identifier from the request message and the trapdoor in each mapping pointer entry as parameters. In this way, the CMS finds the address of the INT mapping system (IMS_D) holding D's semi-masked mapping. Subsequently, the request message is forwarded to the IMS_D.

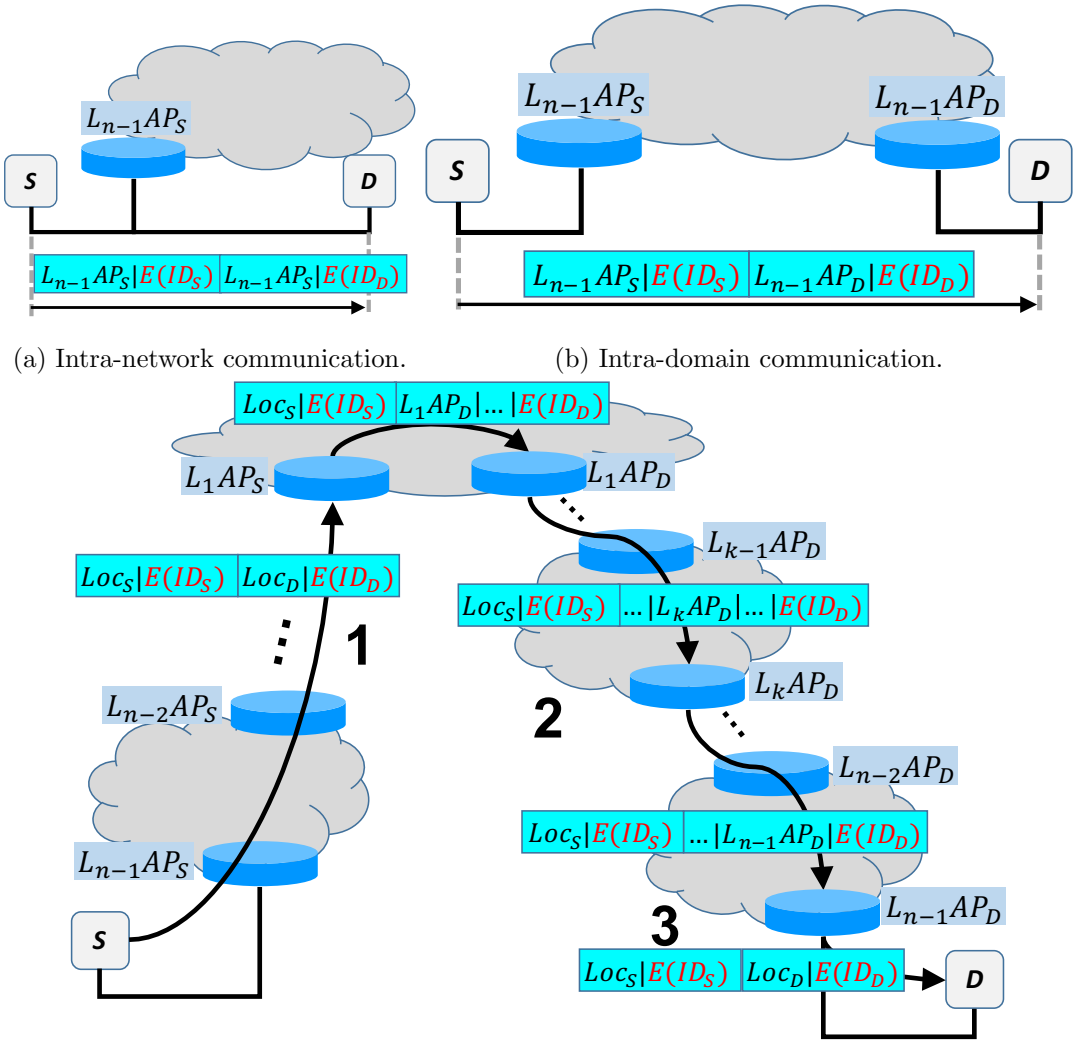
4. After receiving the request, the IMS_D determines D's mapping by performing $Test()$ and responds with D's semi-masked address.

The source endpoint can cache the locator sequence so that it does not have to perform a mapping lookup for each packet. Moreover, the source endpoint, IMS_S, and CMS can use the cleartext addresses of the IMS_S, CMS, and IMS_D, if the destinations of the request message do not have to be masked.

4.2.1.4 Semi-masked packet delivery

The endpoint S wants to send a semi-masked packet to the endpoint D. It is assumed that both endpoints have already registered their semi-masked mappings at the mapping services as described in Section 4.2.1.2. First, the source endpoint encrypts the destination identifier with the public key of the destination endpoint by means of PEKS. After that, S resolves the masked destination identifier to the actual locator sequence of the destination endpoint as introduced in Section 4.2.1.3. After getting D's semi-masked address, the source endpoint generates the packet $< smAddr_D \mid smAddr_S >$ (see Figure 4.31).

- ◇ In case that the destination locator sequence has a length of one, i.e. both endpoints reside in the same INT, the source endpoint puts only its edge-locator into the source locator sequence field. After that, the source endpoint checks whether the destination endpoint possesses the same edge-locator as itself.
  - ◇ If both endpoints have the same edge-locator, the endpoints are connected to the same Edge (see Figure 4.31a). In this case, the source endpoint resolves the masked destination identifier to a MAC address and forwards the packet to the destination endpoint as explained in Section 4.1.1.1.
  - ◇ Otherwise, the packet is locally forwarded on the basis of D's edge locator $L_{n-1}AP_D$ (see Figure 4.31b). After receiving the packet, the destination edge node resolves the masked destination identifier of the packet to a MAC address and forwards the packet to this MAC address.
- ◇ In case of inter-domain communication, i.e. the length of the destination locator sequence is more than one, the semi-masked packet delivery consists of three steps as defined in HAIR (see Figure 4.31c):



(a) Intra-network communication.

(b) Intra-domain communication.

(c) Inter-domain communication.

Figure 4.31: Semi-masked packet delivery in BPF-HAIR.

1. The packet is forwarded up to the Core either by using the default route or by utilising Loc_S in the semi-masked source address.

2. The packet is forwarded domain-wise so that $L_{k-1}AP_D$ acts as an ingress node (and L_kAP_D as an egress node) of the destination routing domain at level k with $2 \leq k \leq n-1$. Here, L_kAP_D is taken as the basis locator for packet forwarding in a routing domain at level k.

3. Eventually, the packet arrives at $L_{n-1}AP_D$ which resolves D's masked identifier to its MAC address. Finally, the packet is forwarded to D's MAC address.

4.2.2 Fully blind packet forwarding

We mask both identifiers and locators in this BPF-HAIR mode. Thus, the fully masked address of the endpoint X with the identifier ID_X is

$$fmAddr_X : \underbrace{E(L_1AP_X)|...|E(L_{n-1}AP_X)}_{mLoc_X} \mid E(ID_X),\ where \tag{4.12}$$

- $E(ID_X) = PEKS(X_{pub}, ID_X)$ is the masked identifier of the endpoint X, which is encrypted with its own public key.
- $E(L_kAP_X) = PEKS((L_kAP_X)_{pub}, L_kAP_X)$ is the masked locator of L_kAP_X, which is encrypted with its own public key.

A fully masked packet from the source endpoint S to the destination endpoint D has the following address fields:

$$fmAddr_D \mid fmAddr_S \mid C(ID_S). \tag{4.13}$$

Here, $fmAddr_D$ and $fmAddr_S$ are D's and S's fully masked addresses generated according to equation 4.12. Moreover, the source identifier ID_S is additionally encrypted with the corresponding public key of D by means of a conventional encryption, e.g., RSA. The last two address fields in a fully masked packet serve the same purposes as in the semi-blind packet forwarding.

Since the locators are masked in this mode, the routing table setup has to be redesigned. For that, we adopt the masked routing approach from BPF-GLI (see Section 4.1.2.1). However, the masked routing is performed domain-wise so that a network node in a domain maintains masked routing information only for the nodes in the same domain. Moreover, a LAP holds at least two masked routing tables for the domains connected with each other by this LAP. In order that the source endpoint requires only the public key of the destination endpoint for fully masked packet generation, the mapping functionalities have to be redesigned accordingly, which we discuss below.

4.2.2.1 Fully blind mapping system

As in the semi-blind mode, this BPF-HAIR mode also adopts HAIR's mapping system architecture. In this mode, both identifiers and locators are encrypted in the CMS and $IMSs$. In the IMS responsible for the INT in which the endpoint X is located, a *fully masked mapping table entry (fm-MTE)* consists of X's masked identifier, trapdoor and actual locator sequence in encrypted form:

$$fm\text{-}MTE_X : [(E(ID_X), T(ID_X)), E(L_1AP_X)|...|E(L_{n-1}AP_X)].$$

Moreover, the IMS keeps the masked locators of LAPs at levels 1 to $n-2$, via which to reach the IMS from the Core. It can be dynamically resolved by running an enhanced masked traceroute, e.g., for the CMS. Here, each LAP puts its masked locator into the message to be sent back to the IMS. Additionally, the DHCP server at an Edge holds the masked locator and trapdoor of the edge node. Furthermore, the IMS and DHCP server keep the fully masked addresses of the CMS and IMS. If the destinations of the mapping registration and lookup traffic do not have to be masked, these addresses are maintained in cleartext.

The CMS holds a table consisting of mapping pointers to $IMSs$ keeping the actual mappings for endpoints. A *fully masked mapping pointer entry (fm-MPE)* for the endpoint X consists of its masked identifier, trapdoor, and the fully masked address of the IMS which maintains X's actual mapping:

$$fm\text{-}MPE_X : [(E(ID_X), T(ID_X)), fmAddr_{IMS_X}].$$

4.2.2.2 Fully masked mapping registration and lookup

In order to bind the masked identifier of the new endpoint D to its actual masked locator sequence, the following steps are performed as defined in the semi-blind mode (see Figure 4.32):

1. The endpoint D sends the tuple $(E(ID_D), T(ID_D))$ to the DHCP server.

2. The DHCP server responds with edge node's trapdoor $T(L_{n-1}AP_D)$ and the fully masked address of the IMS responsible for the INT in which the endpoint resides (IMS_D). After that, the DHCP server maps the tuple to the masked edge locator $E(L_{n-1}AP_D)$ and sends the fully masked edge-mapping to the IMS_D:

   $$fm\text{-}edge\text{-}map\{(E(ID_D), T(ID_D)) \Rightarrow E(L_{n-1}AP_D)\}.$$

   For the incoming edge-mapping, the IMS_D proceeds in the same manner as in the semi-blind mode in order to create a new entry or to update an already existing one for the endpoint D.

3. If a new entry is created for the endpoint, the IMS_D sends the entire masked locator sequence $E(L_1AP_D)|...|E(L_{n-1}AP_D)$ back to the endpoint D. Moreover, the IMS_D sends the fully masked mapping pointer

   $$fm\text{-}map\text{-}pointer\{(E(ID_D), T(ID_D)) \Rightarrow fmAddr_{IMS_D}\} \text{ to the } CMS.$$

   For the incoming mapping pointer, the CMS works analogously to the semi-blind mode.

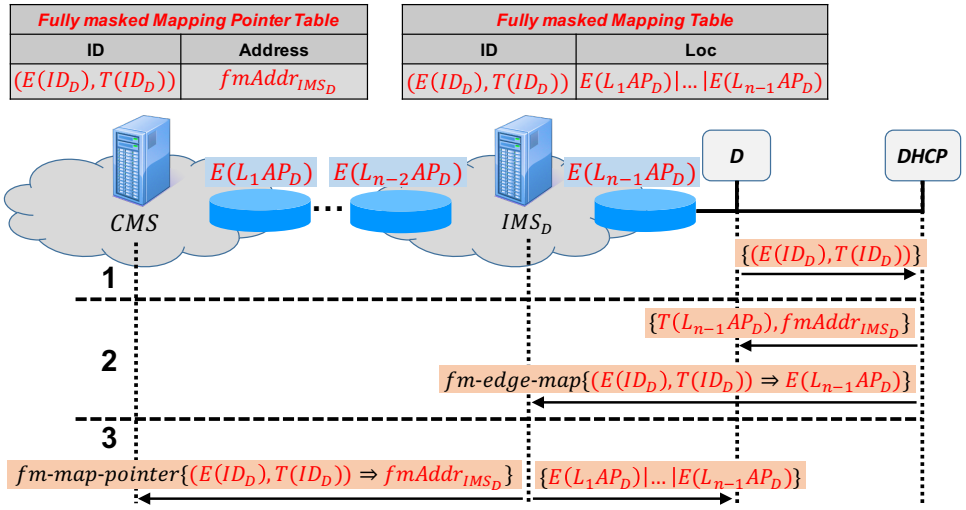
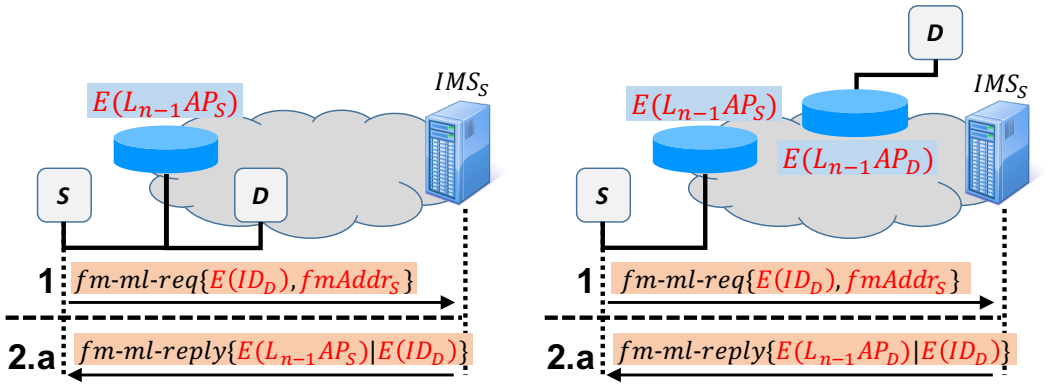
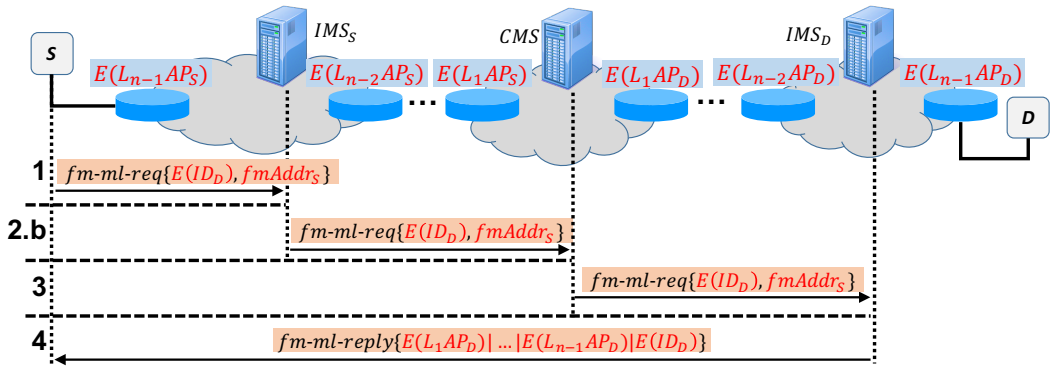


Figure 4.32: Fully masked mapping registration in BPF-HAIR.



(a) Intra-network communication.

(b) Intra-domain communication.

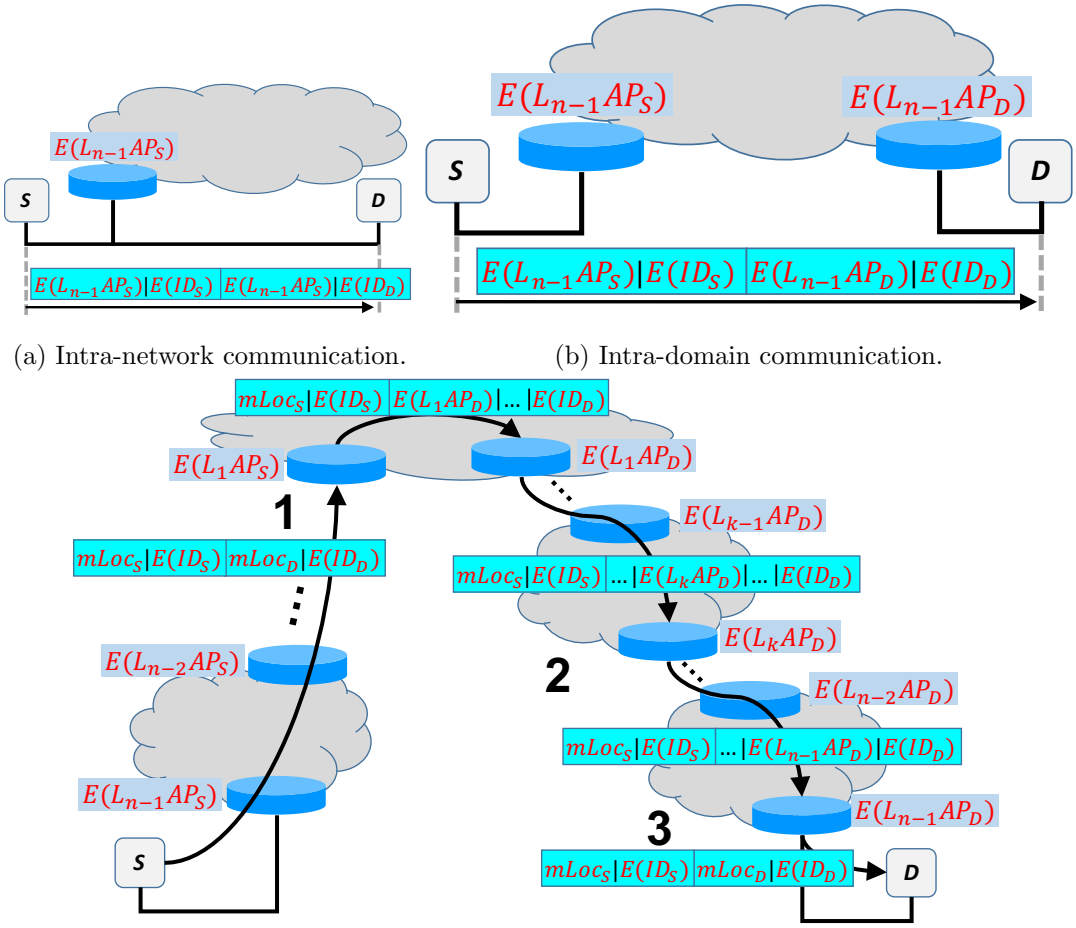


(c) Inter-domain communication.

Figure 4.33: Fully masked mapping lookup in BPF-HAIR.

If the destination addresses of the edge-mapping and mapping pointer messages do not have to be masked, the cleartext addresses of the IMS_D and CMS can be used as proposed in the semi-blind mode. In contrast to the semi-blind mode, the mapping pointer of the endpoint may not contain the cleartext or semi-masked address of the IMS_D, since it can thus reveal information about the INT in which the endpoint resides.

The fully masked lookup is analogous to the semi-masked one except that the source endpoint S gets a fully masked address containing the actual masked locator sequence $mLoc_D$ of the destination endpoint D (see Figure 4.33). Additionally, S sets its own fully masked address as the source address of the request message. Moreover, the cleartext addresses of the mapping services can be used, if the destination addresses of the mapping lookup request do not have to be masked.



(a) Intra-network communication.

(b) Intra-domain communication.

(c) Inter-domain communication.

Figure 4.34: Fully masked packet delivery in BPF-HAIR.

4.2.2.3 Fully masked packet delivery

We assume that the source and destination endpoints S and D have already registered their fully masked mappings at the mapping services as described in Section 4.2.2.2. After getting D's fully masked address by means of a fully masked mapping lookup as introduced in Section 4.2.2.2, the source endpoint generates the packet $< fmAddr_D \mid fmAddr_S >$ (see Figure 4.34).

- ◇ If the destination address is an edge-address, i.e. the destination locator sequence consists of a single masked locator, namely of $E(L_{n-1}AP_D)$, it means that the destination endpoint resides in the same INT as the source endpoint. In this case, the source endpoint puts only its masked edge-locator into the source locator sequence field. After that, the source endpoint performs

  $$Test(E(L_{n-1}AP_D), T(L_{n-1}AP_S)).$$

  Here, the source endpoint has already got $T(L_{n-1}AP_S)$ during the mapping registration, and this value is the trapdoor for the locator of the edge node responsible for the Edge to which the source endpoint is connected.

  - ◇ If $Test()$ returns 1, i.e. both endpoints are connected to the same Edge (see Figure 4.34a), the source endpoint resolves the masked destination identifier to a MAC address and forwards the packet to this MAC address as described in Section 4.1.1.1.
  - ◇ Otherwise (see Figure 4.34b), the packet is forwarded to the edge node. For the incoming fully masked packet, a network node in the INT performs

    $$Test(E(L_{n-1}AP_D), T(Loc_i))$$

    for each masked routing table entry i. The packet is then forwarded via the port mapped in the entry for which $Test()$ returns 1. Eventually, the packet arrives at the destination edge node which resolves the masked destination identifier to a MAC address and forwards the packet to the destination endpoint.

- ◇ In case that the destination locator sequence consists of more than one masked locator, i.e. the source and destination endpoints reside in different INTs, the fully masked packet is delivered in three steps (see Figure 4.34c). These three steps are analogous to the steps in the semi-blind mode except step 2. In this step, for the incoming packet with $mLoc_D = ...|E(L_kAP_D)|...$, a node in routing domain at level k takes k-th masked locator in the sequence as basis for packet forwarding. Here, the node performs

  $$Test(E(L_kAP_D), T(Loc_i))$$

for each masked routing table entry i. If $Test()$ returns 1 for an entry, the packet is then forwarded via the port mapped in that entry. Otherwise, the node can send a destination unreachable message back to S, or it just drops the packet.

4.2.3 Analysis

In BPF-HAIR, the address of an endpoint consists of its actual locator sequence and its identifier. Here, the locator sequence contains the locators of LAPs which have to be traversed for forwarding a packet to the endpoint. Moreover, the identifier is encrypted with the public key of the endpoint, while the locator of a LAP is encrypted with the public key of the LAP. Thus, it is well-defined, which public key has to be taken for which part of the address in order to mask it. Hence, we can state that BPF-HAIR fulfils the requirement for a hierarchical addressing structure. However, the length of the cleartext/masked locator sequence of an endpoint is proportional to the deepness of its level. Thus, the address fields in a packet alone can use a huge part of the packet size, which we discuss in Section 4.2.6 in detail.

BPF-HAIR defines a system which maintains the mappings of the masked identifiers of the endpoints to their actual masked/cleartext locator sequences. By means of mapping lookup, a source endpoint gets the masked locator sequence of the destination endpoint directly from the mapping system. For a semi- and fully masked packet generation, the source endpoint thus requires only the public key of the destination endpoint. The last masked locator in the locator sequence of an endpoint is maintained by the DHCP server at the Edge to which the endpoint is connected. Moreover, the remaining masked locators are held by the mapping system of the INT to which the Edge belongs. Here, the mapping system leverages an enhanced traceroute in order to get these masked locators. Thus, BPF-HAIR does not require an additional infrastructure for exchanging and certifying the public keys of LAPs.

BPF-HAIR's hierarchical scheme reflecting the current Internet structure interprets the entire infrastructure on the basis of multiple levels. Here, the Edges at level n are aggregated to INTs at level $n-1$. Moreover, INTs at level k aggregate into INTs at level $k-1$. In addition, INTs at level 2 are connected to each other via the Core. Thus, the entire infrastructure is accordingly super- and subnetted by design so that an additional process is not required to aggregate and partition the networks. This means that routing table entries in the semi- as well as fully blind modes are already aggregated accordingly. However, this scheme does not define attachment of endpoints to nodes at an arbitrary level. In summary, we can state that BPF-HAIR fulfils the architectural requirements for an adequate BPF design, while the length of a locator sequence for an endpoint is dependent on the level of the domain in which the endpoint is located.

BPF-HAIR defines two modes of blindness, which provide the same NAC levels and unlinkability properties as in BPF-GLI (see Section 4.1.4.1 and 4.1.4.2). In BPF-HAIR, the blindness mode of packet addresses cannot, however, alternate during the transmission. Moreover, BPF-HAIR also allows asymmetric masking of the source and destination addresses of a packet. Thus, a packet can be masked by means of up to four different masking combinations which support various NAC levels, as already discussed in Section 4.1.4.4.

4.2.4 Implementation

In the implementation of BPF-HAIR, two data structures realise semi- and fully masked addresses. Here, the cleartext/masked locator sequence is implemented as a vector of arbitrary length so that we can realise an infrastructure with an arbitrary number of levels. In a cleartext locator sequence, each vector entry is a byte array, while a vector entry in a masked locator sequence is a *Masked_Data* consisting of two byte arrays (see Figure 3.9). The data structure implementing a cleartext/masked locator sequence contains a further attribute which specifies the actual cleartext/masked locator to be taken for packet forwarding.

Beside the semi- or fully masked addresses of the source and destination endpoints, the network packet header structure for BPF-HAIR contains two flags which define the masking types of the communicating endpoints (see Figure 4.35). By means of a further flag in the header structure, it is stated whether the packet has to be forwarded using the default route. In our implementation, we have omitted the third address field as is the case in the implementation of BPF-GLI.

We have implemented multiple packet structures to realise the BPF-HAIR functionalities such as mapping registration and lookup. For the link layer discovery, masked identifier resolution and unmasked/masked routing, the same packet structures have been used as in the implementation of BPF-GLI. For encryption of locators and identifiers as well as for generation of key pairs and trapdoors, we have leveraged the PEKS library [ALPK07].

The BPF-HAIR functionalities on the host side have been realised by *BPF-HAIR-BNS* implemented as a C++ singleton class in the same manner as *BPF-GLI-BNS*. The framework *BPF-HAIR-BNS* interacts with applications via the interface *BPF_HAIR_BNS_Socket* operating in the same way as *BPF_GLI_BNS_Socket*.

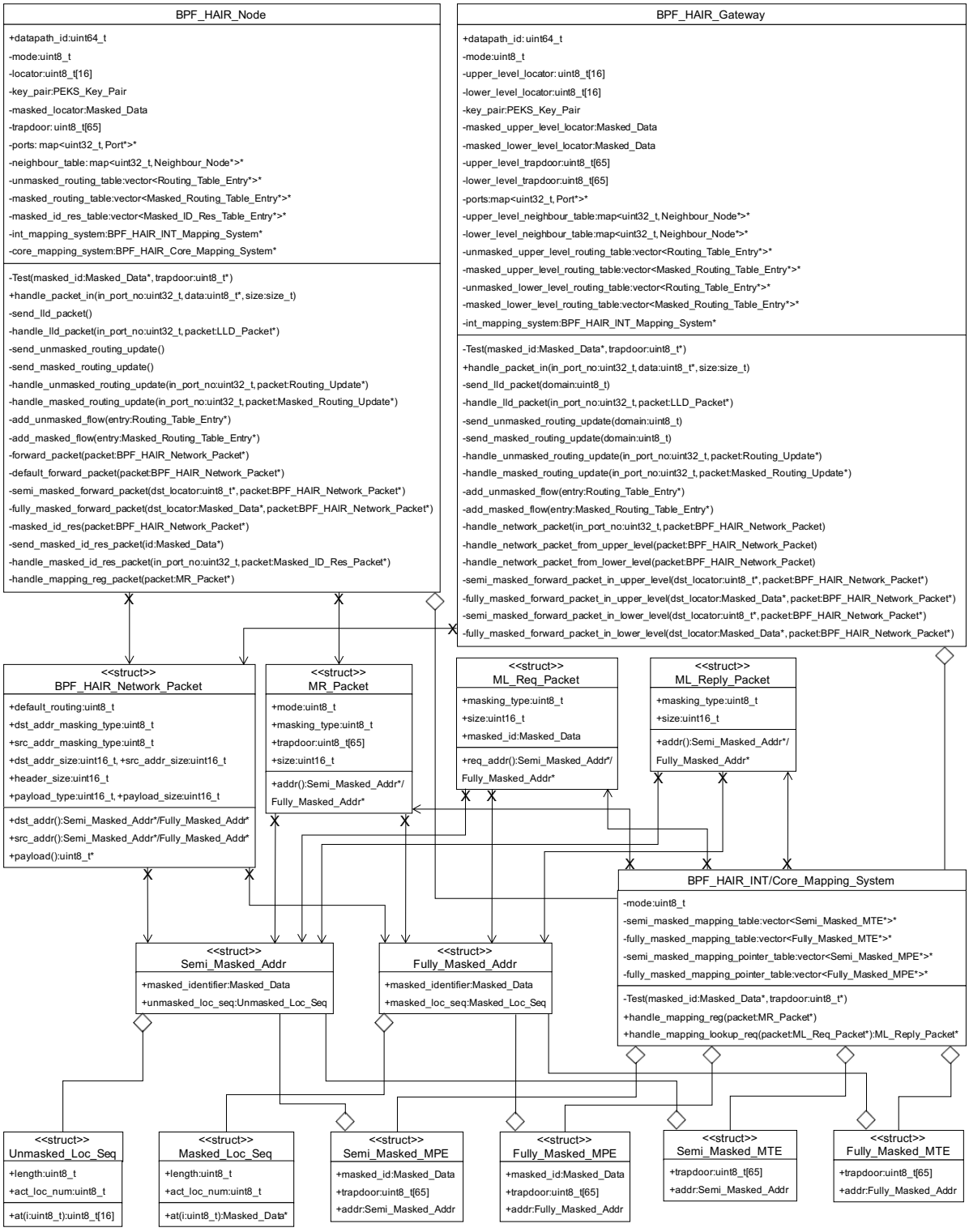


Figure 4.35: UML diagram of *BPF_HAIR_Node* and *BPF_HAIR_Gateway*.

4.2.4.1 Network side

The NOX component *BPF_HAIR* realises the BPF-HAIR functionalities on the network side. Here, the C++ classes *BPF_HAIR_Node* and *BPF_HAIR_Gateway* implement the functionalities of a network node and LAP. An UML diagram of both classes and the associated data structs is given in Figure 4.35. Here, we have omitted the data structures whose UML diagram is already presented in Figure 3.9 and 4.12.

The entire infrastructure in BPF-HAIR is split into the Core and multiple INTs. While the Core is handled as the current Internet backbone, each INT is under the control of a single provider as is the case with ISPs. In the realisation of BPF-HAIR, the network nodes and LAPs of each INT are thus managed by the controller component started for that INT. For each network node and LAP in an INT, a new object is created from the classes *BPF_HAIR_Node* and *BPF_HAIR_Gateway*. Moreover, the controller component for that INT maintains a list of these objects which are managed and identified in the same manner as in the implementation of BPF-GLI.

The routing, mapping and DHCP functionalities in BPF-HAIR have been implemented in the SDN-like manner as already introduced in Section 4.1.5.1. Thus, the routing table setup in each INT takes place at the controller. Moreover, each controller component managing an INT at level $n-1$ creates an instance of the C++ class *BPF_HAIR_INT_Mapping_System* which realises the IMS functionalities. Each object representing a node and LAP in an INT at level $n-1$ holds a pointer to this mapping system instance maintaining a sequence of cleartext and masked locators via which the Core can be reached. Additionally, the class *BPF_HAIR_Node* also implements the functionalities of the DHCP server. Furthermore, our implementation also leverages the flow-based forwarding in the same way as proposed to realise BPF-GLI. After routing table setup, the OpenFlow datapaths thus hold flow tables reflecting the routing tables of network nodes and LAPs in BPF-HAIR.

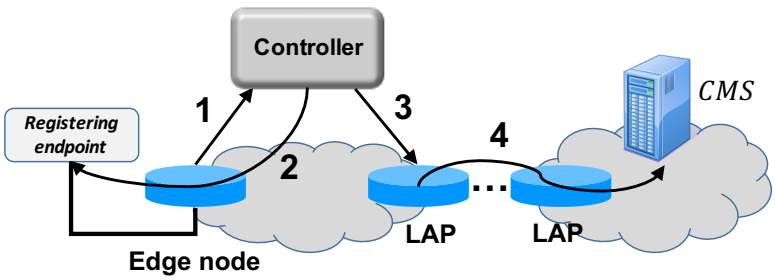


Figure 4.36: SDN-like implementation of mapping registration in BPF-HAIR.

Mapping registration and lookup

A mapping registration packet received by an edge node is handled as follows (see Figure 4.36):

1. The packet matches the flow defined to send mapping registration messages to the controller. Thus, the packet is sent to the controller.
2. The mapping system instance creates a new mapping table entry or updates the already existing one for the registering endpoint. In case of a new entry, the object for the edge node gets the cleartext/masked locator sequence from the mapping system instance and asks the associated OpenFlow datapath to send the sequence back to the requesting endpoint.
3. If a new mapping table entry is created for the endpoint, the controller component orders an OpenFlow datapath acting as a LAP in the INT to send the mapping pointer for the endpoint to the CMS.
4. The message containing the mapping pointer is conventionally forwarded via the OpenFlow datapaths on the route to the CMS.

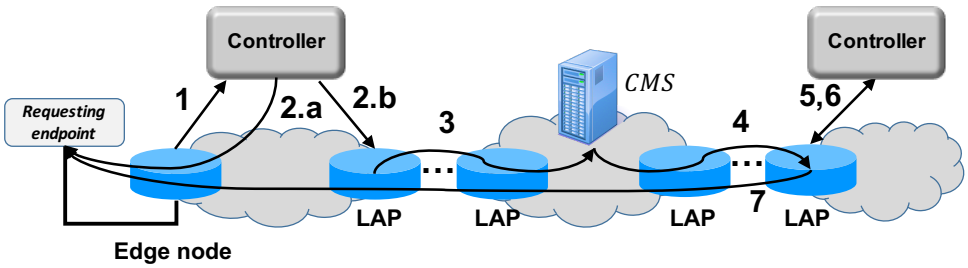


Figure 4.37: SDN-like implementation of mapping lookup in BPF-HAIR.

For a mapping lookup request message arriving at an edge node, it is proceeded as follows (see Figure 4.37):

1. The edge node sends the request message to the controller.
2. The function *handle_mapping_lookup_req_packet()* is called on the mapping system instance.
   (a) If a mapping entry is found for the requested identifier, the edge node object tells the associated OpenFlow datapath to send the mapping lookup reply to the requesting endpoint.
   (b) Otherwise, the controller component instructs one of the OpenFlow datapaths acting as LAPs in the INT to forward the mapping lookup request to the CMS.

3. The request message is conventionally forwarded via the OpenFlow datapaths on the route to the CMS.

4. After the CMS determines the IMS holding the actual locator sequence for the requested identifier, the mapping lookup request is conventionally forwarded to that IMS.

5. If the request arrives at a LAP responsible for the INT in which the destination IMS resides, the OpenFlow datapath representing this LAP sends the request to the controller.

6. After finding the mapping table entry for the requested identifier, the object representing the LAP tells the associated OpenFlow datapath to send the mapping lookup reply back to the requesting endpoint.

7. The reply message is forwarded via the OpenFlow datapaths on the route to the requesting endpoint.

In this way, an endpoint sending a mapping lookup request message does not need the address of the IMS responsible for the INT in which the endpoint resides. However, the reply message is conventionally forwarded via the OpenFlow datapaths on the route between the requesting endpoint and the IMS holding the actual mapping for the requested identifier. Hence, the endpoint has still to put its own address into the mapping lookup request message as the address of the requesting endpoint. The implementation of the CMS and the handling of mapping registration and lookup request messages at the CMS are given in Section 4.2.5

Intra-domain communication

If the source and destination endpoints are connected to Edges in the same INT, it is proceeded as given in the implementation of BPF-GLI. Thus, the packet is sent to the controller at the source edge in order to add a conventional IPv4 header. Until the packet arrives at the destination edge node, the packet is forwarded by means of the flows defined at the nodes on the route. Upon receiving the packet, the destination edge node sends it to the controller in order to remove the IPv4 header and to perform masked identifier resolution.

Inter-domain communication

If the communicating endpoints reside in different INTs, a BPF-HAIR network packet is handled as follows (see Figure 4.38):

1. Since the packet matches the flow defined to send BPF-HAIR network packets to the controller, the source edge node sends the packet to the controller. The object representing the source edge node at the controller checks whether the flag for default routing is set. Since this is the case here, a conventional IPv4 header is added to the packet,

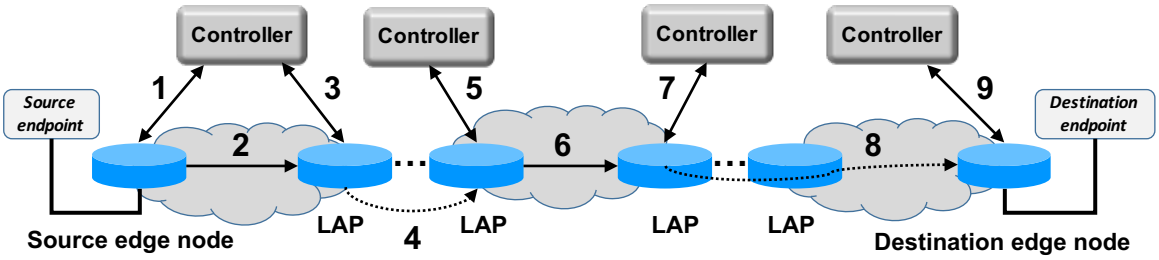


Figure 4.38: Flow-based packet forwarding in inter-domain communication.

and the last four bytes of the unmasked and masked locator of the next LAP for default routing are written into the IPv4-SRC field in the semi- and fully blind mode. Thus, the packet becomes an IPv4 packet. The object orders the associated OpenFlow datapath to forward the updated packet via the specified port.

2. Until the packet arrives at a LAP, it is forwarded using the flows at the nodes on the route.

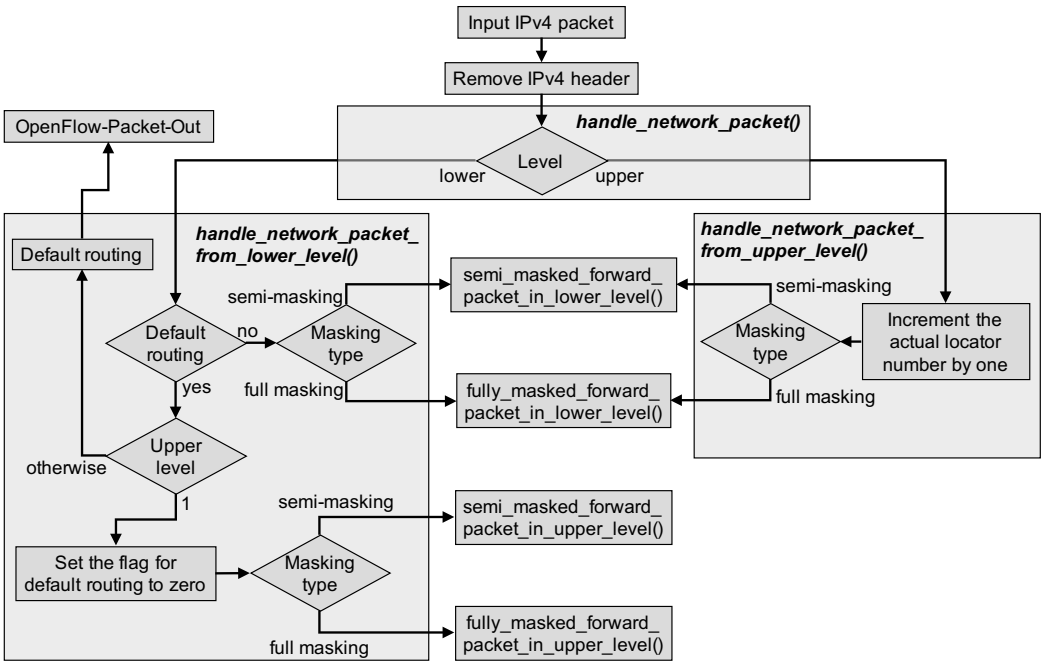


Figure 4.39: Flowchart for packet handling at LAP.

3. If a LAP receives the packet, it sends the packet to the controller component which calls the function *handle_network_packet()* on the object representing the LAP (see flowchart in Figure 4.39). After determining that the packet comes from the INT at the lower

level, the function *handle_network_packet_from_lower_level()* checks whether the flag for default routing is set.

- ◇ If this is not the case, the packet is forwarded back within the INT at the lower level according to the masking type of the destination address.
- ◇ Otherwise, the upper level of the LAP is checked.
  - ◇ If the LAP connects an INT at level 2 with the Core, step 5 is performed.
  - ◇ Otherwise, the controller component writes the last four bytes of the unmasked and masked locator of the next LAP for default routing into the IPv4-SRC field in the semi- and fully blind mode. After that, the associated OpenFlow datapath is asked to forward the updated packet via the specified port.

4. The steps 2 and 3 are repeated until the packet arrives at a LAP connecting an INT at level 2 with the Core.

5. If the packet arrives at an LAP connecting an INT at level 2 with the Core, the object representing the LAP checks the destination address masking type after setting the flag for default routing to zero.

   - ◇ In case of semi-masking, the function *semi_masked_forward_packet_in_upper_level()* is called. This function takes the actual unmasked locator (i.e., the first unmasked locator) from the destination locator sequence to perform the conventional longest prefix matching. After that, the function writes the last four bytes of the locator mapped in the matched entry into the IPv4-SRC field.
   - ◇ The function *fully_masked_forward_packet_in_upper_level()* is called, if the destination address is fully masked. Here, the actual masked locator (i.e., the first masked locator) from the destination locator sequence is taken to perform $Test()$, and the value in the IPv4-SRC field is updated accordingly.

   The LAP object orders the associated OpenFlow datapath to forward the packet within the Core.

6. From then on, the packet is flow-based forwarded within the Core.

7. For the incoming packet, a LAP sends it to the controller. Since the packet comes from the domain at the upper level, the function *handle_network_packet_from_upper_level()* is called on the object representing the LAP. After incrementing the actual locator number by one, this function first checks the masking type of the destination address.

   - ◇ The function *semi_masked_forward_packet_in_lower_level()* is called, if the destination address is semi-masked. This function takes the actual unmasked locator

from the destination locator sequence and performs the conventional longest prefix matching. After that, the last four bytes of the unmasked locator in the matched entry are written into the IPv4-SRC field.

- ◇ In case of full masking, the function *fully_masked_forward_packet_in_lower_level()* is called. This function performs $Test()$ with the actual masked locator and the trapdoor in each routing table entry as parameters. Thus, the function updates the value in the IPv4-SRC field with the last four bytes of the masked locator mapped in the entry for which $Test()$ returns 1.

The object representing the LAP instructs the associated OpenFlow datapath to forward the updated packet within the domain at the lower level. The packet is then forwarded using the flows at the nodes on the route to the next LAP.

8. Step 7 is repeated until the packet arrives at the destination edge node.

9. Eventually, the destination edge node receives the packet and sends it to the controller. The object representing the edge node removes the IPv4 header from the packet and resolves the masked destination identifier to a MAC address. After that, the object tells the associated OpenFlow datapath to send the updated packet via the specified port.

While the packet is thus forwarded by means of the flows at the nodes between LAPs, the packet is sent to the controller at each LAP in order to update its IPv4-SRC field.

4.2.5 Testbed

To deploy the implementation of BPF-HAIR, we have used the same hardware devices, i.e. two PCs and three HP 3800-24G-2SFP+ switches as in the deployment of BPF-GLI's implementation. Here, 16 OpenFlow instances run on these three HP OpenFlow switches. The OpenFlow instances are interconnected with each other according to the testbed topology consisting of four levels.

In this topology, we have the Core, four INTs, and two Edges. Here, the Core and each INT consist of two nodes. Moreover, the domains are connected with each other by a single LAP as illustrated in Figure 4.40. Thus, each node in each domain maintains four routing table entries. Furthermore, each LAP (except L_3AP_S and L_3AP_D) has two routing tables holding four entries for each level. In this topology, the communicating endpoints S and D are connected to the edge nodes belonging to different INTs. Thus, a packet from S to D, or vice versa, takes 16 hops. For these two endpoints, we have started two instances of BPF-HAIR-BNS running on the PC with an Intel Core2 Duo 3.33 GHz CPU.

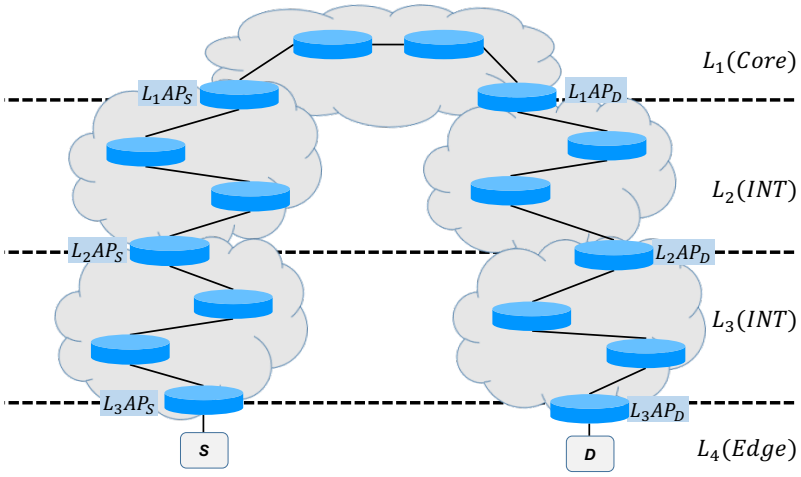


Figure 4.40: Testbed topology for BPF-HAIR.

For each INT, its own controller component instance runs on the PC with an Intel Core 2 Extreme 3.06 GHz CPU. In order to simplify the deployment, two nodes in the Core are managed by a single controller component as is the case in the deployment of BPF-GLI's implementation. Thus, five instances of the controller component run on this PC (see Figure 4.41).

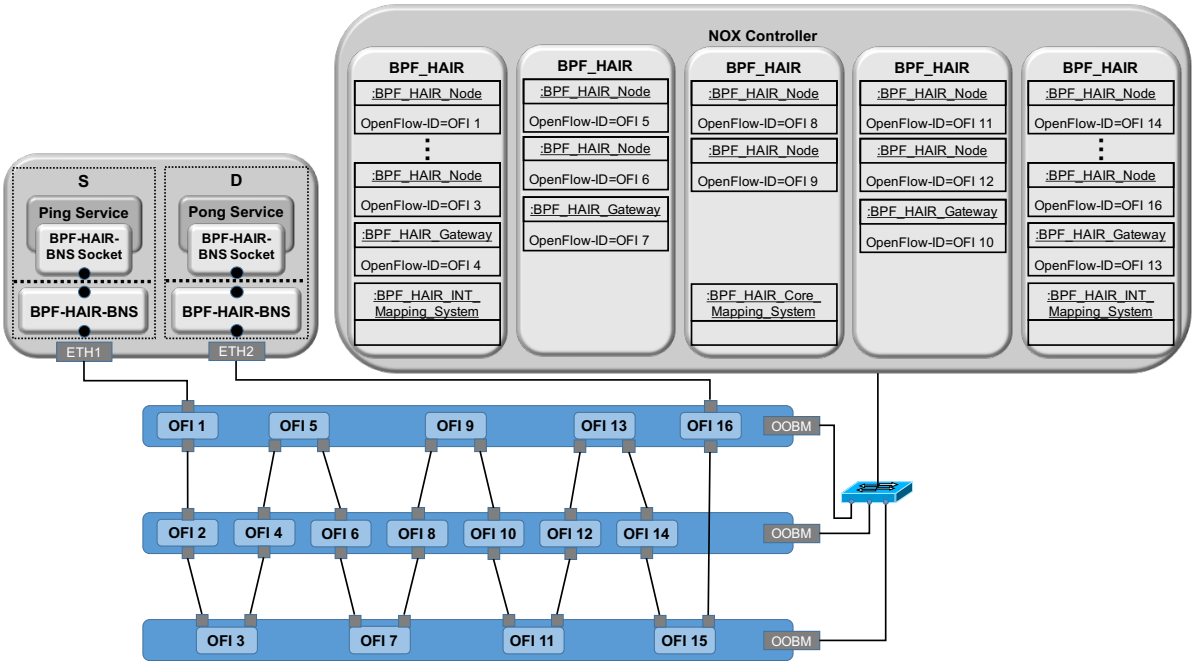


Figure 4.41: OpenFlow testbed for BPF-HAIR.

In our deployment, the core mapping service is realised by an instance of the C++ class *BPF_HAIR_Core_Mapping_System*, which is created by the controller component managing the Core. Here, each node in the Core holds a pointer to this mapping system instance. Moreover, the mapping system object keeps the ID of an OpenFlow datapath acting as a Core node to send packets via this datapath.

For an incoming mapping pointer registration, a Core node sends it to the controller which calls the function *handle_mapping_pointer_reg()* on the mapping system object. If a Core node receives a packet holding a mapping lookup request, it sends the packet to the controller component. Here, the function *handle_mapping_lookup_req()* is called on the mapping system object. After determining the INT mapping service holding the locator sequence for the requested identifier, the mapping system object orders the OpenFlow datapath (whose ID is kept by the object) to send the request to the INT mapping service.

4.2.6 Evaluation

In the implementation of BPF-HAIR, an endpoint and a node has a cleartext identifier and locator of 16 bytes. Thus, the benchmarks for the cryptographic operations on the PC with an Intel Core2 Duo 3.33 GHz CPU are the same ones as in the implementation of BPF-GLI, which are already presented in Table 3.1. For masked identifier resolution and umasked/-masked routing in BPF-HAIR, we have used the same structures as in BPF-GLI's implementation. The sizes of these structures are already given in Table 4.2.

Table 4.4 shows the sizes of unmasked, semi-, and fully masked structures implemented for BPF-HAIR which has been deployed in the topology illustrated in Figure 4.40. In this topology, the address of an endpoint contains three locators. Thus, the address structure and the other structures containing the address structure have a huge size, and their sizes are highly dependent on the number of the levels. Moreover, this effect is more reflected in case of full masking. In summary, BPF-HAIR introduces a size overhead increase by a factor 5.55 and 12.16 in average for semi- and full masking implemented in a topology consisting of four levels.

Table 4.5 presents the convergence times of conventional and SDN-like routing table setups in a domain. By means of the SDN-like realisation, we have also achieved a performance increase here similar to the implementation of BPF-GLI. However, the routing table setup in a domain converges much faster than in BPF-GLI (compare with Table 4.3). The cause of this is that BPF-HAIR splits the entire infrastructure into more fine-grained domains. Thus, BPF-HAIR

	Size in bytes			*Increase by factor*	
	Unmasked	***Semi-masked***	***Fully masked***	***Semi-masked***	***Fully masked***
Address	66	243	774	3.68	11.72
Network header	145	499	1561	3.44	10.76
Edge mapping reg. packet	17	259	259	15.23	15.23
Mapping pointer reg. packet	83	325	1033	3.91	12.44
Mapping lookup req. packet	86	440	971	5.11	11.29
Mapping lookup rep. packet	69	311	842	4.50	12.20
Mapping table entry	70	314	845	4.48	12.07
Mapping pointer table entry	86	330	1037	3.83	12.05
Mapping lookup cache entry	90	527	1058	5.85	11.75

Table 4.4: Size of unmasked, semi-, and fully masked structures.

	Time in milliseconds		*Increase by factor*
	Unmasked	***Masked***	
Conventional routing table setup	15.27	642.94	42.10
SDN-like routing table setup	5.05	495.15	98.04

Table 4.5: Routing convergence times.

nodes have to maintain smaller routing tables than in BPF-GLI. In this regard, we can state that BPF-HAIR's architecture is more scalable than the architecture in BPF-GLI. The execution times for resolving an unmasked and a masked identifier to a MAC address in case of an empty neighbour cache are already given in Table 4.3.

The performance of the controller component and network stack for BPF-HAIR has been evaluated in the same ping-pong scenario as in the evaluation of BPF-GLI (see Section 4.1.7). Figures 4.42 and 4.43 show the individual RTTs for the unmasked, semi-, and fully masked ten ping and pong packets which are handled hop-by-hop and flow-based.

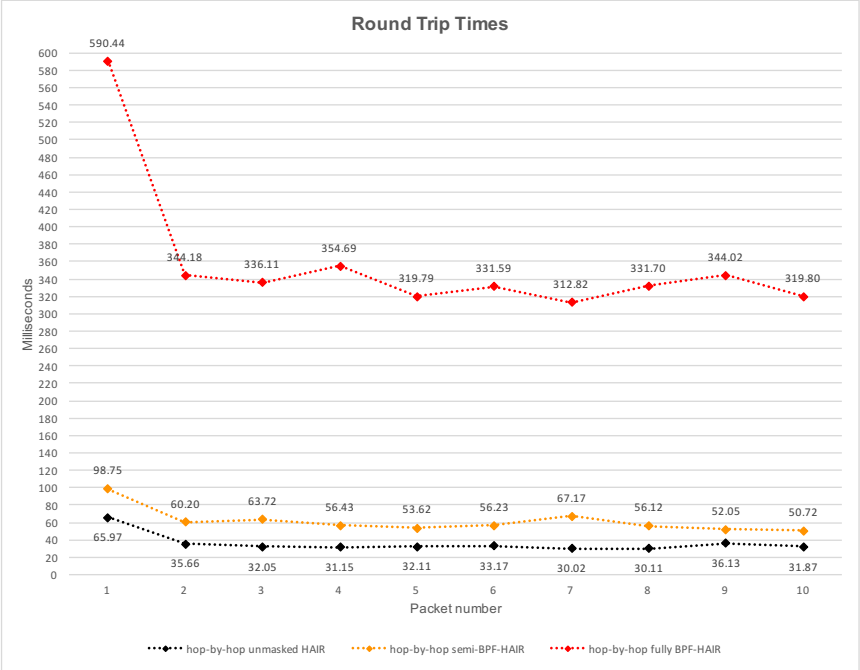


Figure 4.42: Round trip times for packets handled hop-by-hop.

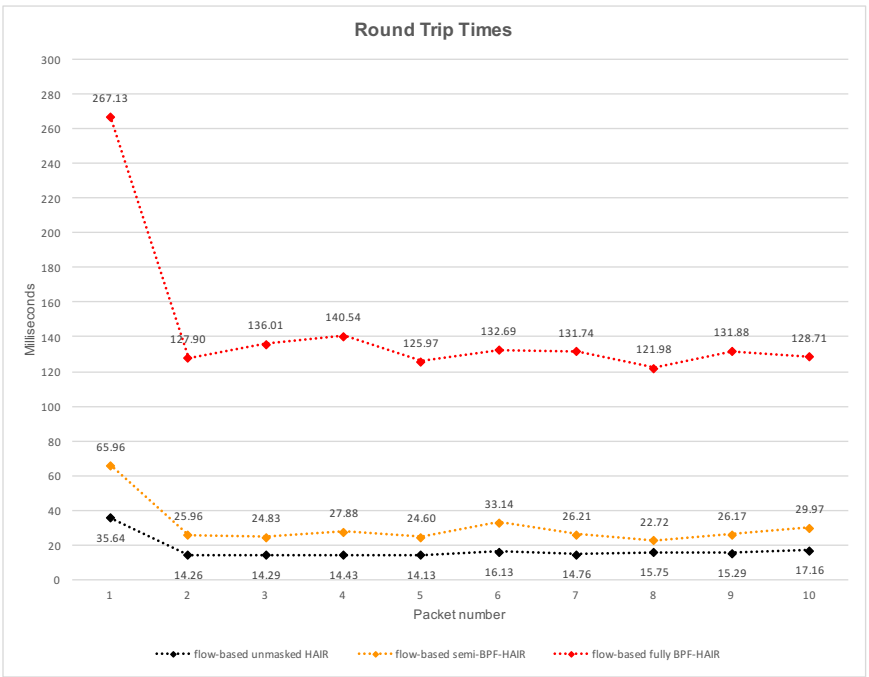


Figure 4.43: Round trip times for packets handled flow-based.

Since the source endpoint has first to find out the actual locator sequence of the destination endpoint, the RTT for the first packet in each case is more than for the remaining packets. The RTT differences between the first packet and the remaining packets represent the execution times for mapping lookups in each case. Here, the mapping lookups take more time than in BPF-GLI, since the mapping lookup requests are forwarded to the mapping system in the INT in which the destination endpoint resides.

While the unmasked and semi-masked packets have similar RTTs as in BPF-GLI, the round trip for fully masked packets take more time than in BPF-GLI. This is because the packets are sent to the controller at each LAP, and the controller component performs $Test()$ for each incoming packet, which is a costly process. This means that BPF-HAIR with four levels runs more $Test()$ calls than BPF-GLI. Thus, the RTT differences between semi- and fully BPF-HAIR are huge, while these differences in BPF-GLI are smaller. Although flow-based semi-BPF-HAIR providing NIC and its unlinkability properties is almost as performant as unmasked HAIR, fully BPF-HAIR cannot satisfactorily be put into practice due to its size and execution time overheads.

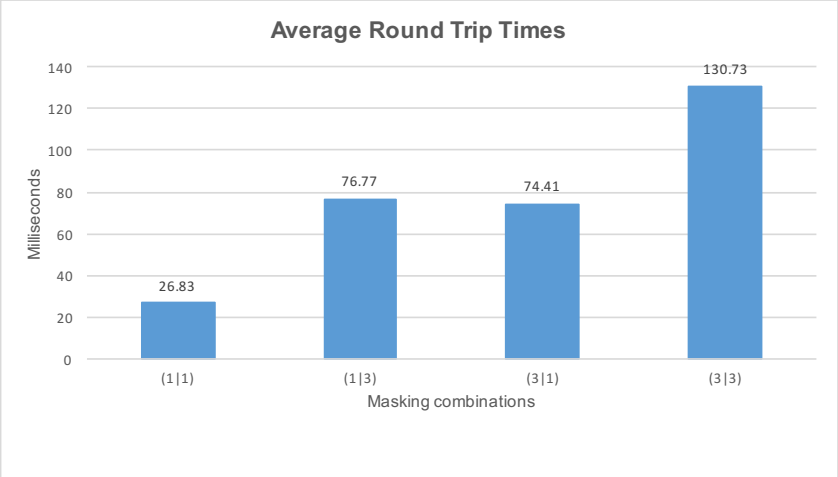


Figure 4.44: Average round trip times for masking combinations.

In BPF-HAIR, the packet addresses can be also masked asymmetrically. Figure 4.44 shows the average RTTs of ten packets to which four masking combinations are applied. Here, every ten packets are handled flow-based. In Figure 4.44, we have used the same denotation as in the evaluation of BPF-GLI (see Section 4.1.7). Thus, 1 denotes the semi-masking, while 3 stands for the full masking. The reversed maskings (1 | 3) and (3 | 1) obtain similar average RTTs

as is the case in BPF-GLI, and both asymmetric maskings achieve smaller RTTs than the symmetric full masking. Instead of symmetric full masking, which introduces huge size and time overheads, both asymmetric maskings could thus be leveraged in case of a client-server communication, where the server is well-known. Here, the client sends packets masked in the combination (1 | 3), and the server responds with packets masked in the combination (3 | 1).

4.3 Conclusion

This chapter has defined the requirements to be fulfilled by a suitable architecture for BPF. In order to achieve the features for an adequate BPF design, we have used the Loc/ID split principle which supports scalability, mobility and multihoming by design and is regarded as the de facto addressing standard for FNA. Here, we have chosen the approaches GLI-Split and HAIR relying on this principle and extended the basic BPF design to BPF-GLI and BPF-HAIR.

BPF-GLI fulfils the architectural requirements with the exception of a fine-grained hierarchical architecture so that performing of the super- and subnetting is not always possible in its entirety. This BPF extension supports three modes of blindness providing different NAC levels which can be flexibly adapted to certain use cases.

While the semi-blindness only supplies NIC, the full blindness provides both NIC and NLC. In case of intra-domain communication, the alternately blind mode only offers NIC as in the semi-blind mode, while NIC and domain-to-domain NLC are supplied in case of inter-domain communication. The sender/recipient and relationship unlinkability properties of these three blindness modes have been discussed against local, weak and strong global adversaries.

In BPF-GLI, it is also possible to mask the source and destination address of a packet in different modes. Thus, BPF-GLI does not only support a symmetric NAC with multiple levels, but also an asymmetric NAC for two communicating endpoints. In this regard, up to nine different masking combinations are possible for a packet between two communicating endpoints residing in different local domains. This means that BPF-GLI provides up to nine different NAC levels. Moreover, endpoints signal their masking modes implicitly so that an additional infrastructure is not needed for signalling the masking modes of the endpoints.

BPF-HAIR fulfils all architectural requirements for an adequate BPF design, while the length of a locator sequence for an endpoint is dependent on the level of the domain in which the endpoint is located. Moreover, attaching endpoints to nodes at an arbitrary level is not defined in BPF-HAIR. This BPF extension also defines semi- and full blindness modes which provide the same NAC levels as in BPF-GLI. In BPF-HAIR, the blindness mode of packet

addresses cannot, however, alternate during the transmission. Moreover, BPF-HAIR also allows asymmetric masking of the source and destination addresses of a packet. Thus, a packet can be masked by means of up to four different masking combinations which support various NAC levels.

This chapter has also presented the prototype implementations of both BPF extensions. For the implementations on the network side, we have utilised the SDN protocol OpenFlow 1.3 by expanding the OpenFlow controller NOX by two further components. Here, we have leveraged the SDN-like realisation and flow-based forwarding for the implementations of BPF-GLI and BPF-HAIR. On the host side, we have extended the framework BNS to BPF-GLI-BNS and BPF-HAIR-BNS. The prototype implementations of both BPF extensions have been deployed in a real hardware testbed to demonstrate their feasibilities for practical deployment capability.

In our evaluations, we have compared the sizes of the unmasked, semi-, and fully masked structures. Moreover, we have evaluated the convergence times for the unmasked and masked routing table setups which are implemented conventionally and in the SDN-like manner. Here, it has been demonstrated that the SDN-like realisation can achieve a not inconsiderable performance increase for the routing table setup. In addition, the performances of the controller components and the extended blind network stacks have been evaluated in a ping-pong scenario. These basic experiments have demonstrated that the flow-based forwarding can obtain a substantial improvement of packet round trip times in comparison with the hop-by-hop handling of packets. In two further experiments, we have evaluated the average RTTs of ten packets that are transferred flow-based between the communicating endpoints. Here, nine masking combinations from BPF-GLI and four masking combinations from BPF-HAIR have been applied to every ten packets. In both experiments, it has been stated that the stronger the NAC level applied to the packets, the higher is the average RTT for these packets.

In summary, we can state that BPF-GLI and BPF-HAIR demonstrate that BPF can principally be realised by means of the existing FNA approaches and introduce significant improvements on the security with tolerable overheads. However, each of both BPF extensions cannot alone achieve all of the properties required for an adequate BPF design. In the next chapter, both BPF designs are thus combined in a way that we only adopt the beneficial aspects of both designs.

Chapter 5

BPF in hierarchical level-based Loc/ID Split

The basic Blind Packet Forwarding (BPF) design demonstrates the general feasibility to simultaneously establish the packet forwarding service and to provide network address confidentiality (NAC) and its unlinkability properties. However, this design is not suitable to be deployed in the current Internet architecture. Therefore, we have defined requirements to be fulfilled by an adequate BPF design. In order to achieve the features for a suitable BPF architecture, the basic BPF design has been extended to the Blind Packet Forwarding in Global Locator, Local Locator, and Identifier Split (BPF-GLI) and to the Blind Packet Forwarding in Hierarchical Architecture for Internet Routing (BPF-HAIR). Both extensions rely on the Loc/ID split principle which is regarded as the de facto addressing standard for Future Network Architecture (FNA).

BPF-GLI fulfils the architectural requirements with the exception of a fine-grained hierarchical architecture so that performing the conventional super- and subnetting is not possible in the fully blind mode. Moreover, BPF-GLI enables that the blindness modes of packet addresses can alternate at domain borders due to locator substitutions on gateways. BPF-HAIR fulfils all architectural requirements for an adequate BPF design, while it, however, introduces its own issues. Since BPF-HAIR forwards packets by means of a loose source routing, the locator sequence of an endpoint address defines the Level Attachment Points (LAPs) to be traversed for forwarding a packet to the endpoint. Thus, the deeper the level, the longer are the locator sequences of a packet. Hence, the address fields in a packet use a huge part of the packet size, and their sizes are highly dependent on the number of the levels. Moreover, attaching endpoints to nodes at an arbitrary level is not defined in BPF-HAIR. Furthermore, the blindness modes of packet addresses cannot alternate because of the loose source routing in BPF-HAIR.

This chapter combines the beneficial aspects of both BPF extensions into a new design, which we call *Blind Packet Forwarding in Hierarchical Level-based Locator/Identifier Split (BPF-HiLLIS)*. It does not only fulfil the requirements for an adequate BPF design but also introduces a fine-grained, flexible and dynamic blindness. In BPF-HiLLIS, we adopt the hierarchical scheme from BPF-HAIR and extend it such that endpoints can be attached to nodes at an arbitrary level. Instead of the loose source routing, LAPs substitute upper and lower level locators with each other for incoming and outgoing packets as is the case in BPF-GLI. Thus, BPF-HiLLIS allows to flexibly alternate from the semi- to the fully blind mode, or vice versa, at domain crossings at arbitrary levels.

In BPF-HiLLIS, endpoints can flexibly state their masking requests by defining *masking vectors*. Masking vectors in a packet determine which blindness mode has to be applied to the packet addresses at which level. The blindness level provided by a masking vector is specified by a *masking rank*. Thus, up to 2^k masking ranks can be established for the address of an endpoint connected to an edge node at level k. Additionally, the source and destination addresses of a packet can be masked asymmetrically. Thus, up to 2^{s+d} masking rank combinations are possible for the addresses of a packet from a source endpoint at level s to a destination endpoint at level d. Here, certain masking ranks providing different NAC levels are classified into two blindness taxonomies having different purposes.

If both the semi- and the fully blind mode in BPF-GLI and BPF-HAIR are applied in a domain, i.e nodes in the domain maintain both the unmasked and the masked routing tables, the nodes can make void the effect of the full blindness by exploiting their unmasked routing tables. This chapter resolves this issue and makes it possible to apply both blindness modes in a domain in a way that the full blindness is still effective. Moreover, the fully blind mode in BPF-GLI and BPF-HAIR requires to set up and maintain masked routing tables in entire domain, which is a costly process as evaluated in Sections 4.1.7 and 4.2.6. In principle, only network nodes on the route between two communicating endpoints need to maintain according masked routing table entries to enable the fully blind packet forwarding for these two endpoints. This chapter also presents an approach which enables a selective masked routing table entry setup so that the fully blind packet forwarding can be performed on demand.

In the implementations of BPF-GLI and BPF-HAIR, we have reinterpreted one of the already existing flow match field types in order to be able to define flows which make it possible that packets do not have to be handled hop-by-hop by the controller. In our evaluations, it has been demonstrated that we can obtain a substantial improvement on packet round trip

times in this way. However, packets still have to be sent to the controller at certain nodes, since the flows, which we could define, are not able to realise the required operations at these nodes. For the implementations of our clean-slate architectures, we thus could not leverage the benefits of OpenFlow in its entirety.

In principle, it is generally possible to extend OpenFlow accordingly by flow match field types required by clean-slate architectures. But the extensions would be highly architecture specific, and each extension would introduce huge implementation costs. Essentially, the main reason why clean-slate approaches cannot leverage OpenFlow in its entirety is that OpenFlow is based on TLV format, and the types defined in OpenFlow rely on the IP packet structure. In this regard, we replace the TLV-based mechanism in OpenFlow with an *Offset-Length-Value (OLV)*-based proceeding, which we call *OLV-OpenFlow*. By means of this OpenFlow version, we intend that not each packet has to be sent to the controller at certain nodes, but rather only the first packet. On the basis of the first packet, the required flows can be set up for two communicating endpoints so that the remaining packets can be forwarded flow-based without sending them to the controller.

After describing the basic idea of BPF-HiLLIS in Section 5.1, its formal construction is presented in Section 5.2. In Section 5.3, we discuss the architectural features of BPF-HiLLIS as well as the NAC levels and their unlinkability properties provided by the masking ranks which are classified into two blindness taxonomies. After introducing the construction and implementation of OLV-OpenFlow in Section 5.4, we present BPF-HiLLIS's implementation and its deployment in a software testbed in Sections 5.5 and 5.6. Finally, we evaluate the implementation in Section 5.7.

5.1 Basic idea

BPF-HiLLIS combines BPF-HAIR's hierarchical scheme and BPF-GLI's locator-substitution on gateways with each other. Instead of either semi- or full masking at all levels, it is thus possible to mask packet addresses in the semi- and fully blind modes level by level. In this way, endpoints can flexibly define blindness scopes with regard to security properties (e.g., private/public, or protected/insecure) of domains in which they are located or via which to reach them. In BPF-HiLLIS, a masking vector for a packet address defines which masking mode has to be applied at which level. A masking vector is realised as a bit vector in big-endian order. The first most-significant bit is for level 1, the second one is for level 2, and so on. Here, "0" stands for semi-masking, and "1" stands for full masking. For example, if the first bit of a masking vector for a packet address is equal to 1, the address is fully masked at level 1.

The blindness level provided by a masking vector for a packet address is represented by a natural number called masking rank. Masking ranks are assigned to PEKS public keys used for encrypting the identifier part of endpoint addresses (see Section 5.2.2). A masking rank specifies how many bits have to be set in the associated masking vector. Thus, the more bits that are set in a masking vector, the higher blindness level provided by the masking vector. Here, multiple taxonomies can be defined to specify the order for the bit-setting, e.g., beginning with the most-significant or most-least bit. Figure 5.1 illustrates both taxonomies used by an endpoint being located at level 3.

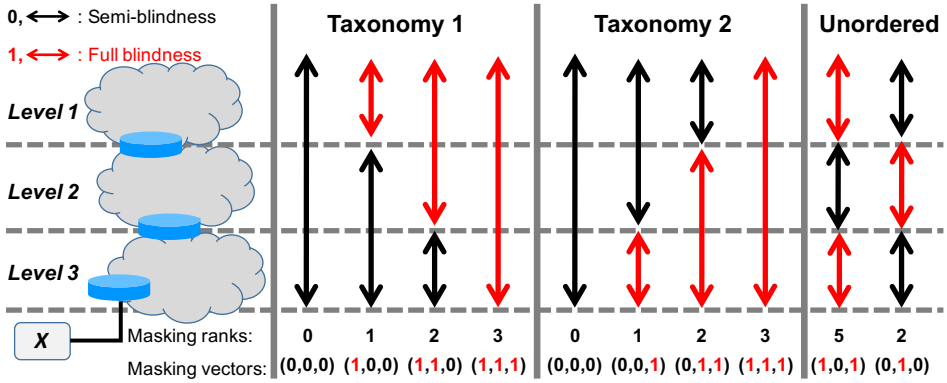


Figure 5.1: Masking ranks and vectors for an endpoint at level 3.

By applying higher masking ranks in the first taxonomy, the scope of the full blindness for the endpoint begins within the Core and expands in the direction of the endpoint. In a semi-blind domain, the endpoint reveals its relative location and can thus be vulnerable to active attacks performed against the entire domain, e.g., manipulating all packets originating from a certain domain. In this regard, the endpoint can select the semi-blindness within a domain, if the domain is protected against such attacks. Hence, the radius of domains regarded as secure becomes smaller by applying higher masking ranks in this taxonomy. On that score, this taxonomy is comparable with the security taxonomy in IPsec-VPN [YS01], where IP Encapsulating Security Payload (ESP) tunnel length becomes longer.

In contrast to the first taxonomy, the endpoint can also begin with the most-least bit (see Figure 5.1). By applying higher masking ranks in the second taxonomy, the radius of domains becoming fully blind grows in the direction of the Core, beginning with the domain in which the endpoint is located. In this way, the endpoint can mask its respective location and thus protect itself against domain-based attacks within higher domains.

BPF-HiLLIS also enables that the endpoint can freely select semi- or full masking at an

arbitrary level (see Figure 5.1). In this case, a masking rank is the decimal value of the binary number represented by the associated masking vector, e.g., $5 \equiv (1,0,1)$, or $2 \equiv (0,1,0)$. In this way, the endpoint can select full masking only within the intermediate domain regarded as unprotected by applying the masking vector $(0,1,0)$. The endpoint can perform the selection either implicitly during the mapping registration (see Section 5.2.3) or on demand by a masking setup request (see Section 5.2.8).

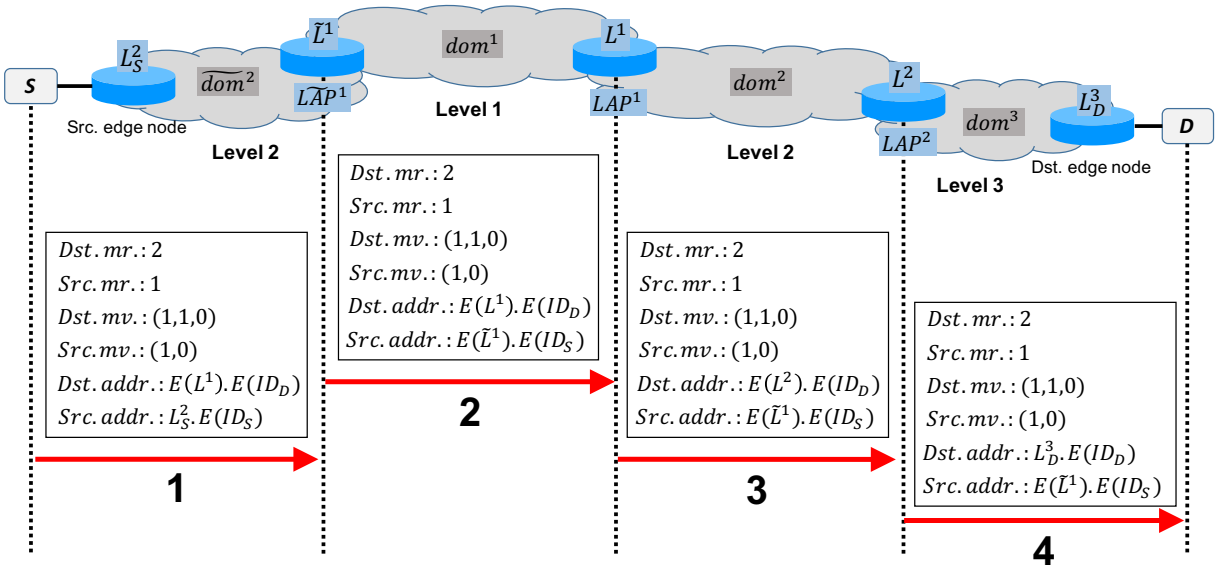


Figure 5.2: Example scenario.

We describe the basic idea of BPF-HiLLIS by means of the following scenario (see Figure 5.2). The topology in our scenario consists of a domain at level 1 (dom^1), two domains at level 2 ($\widetilde{dom}^2$ and dom^2), and a domain at level 3 (dom^3). The level attachment points (gateways) $\widetilde{LAP}^1$ and LAP^1 connect $\widetilde{dom}^2$ and dom^2 to dom^1, and dom^3 is connected to dom^2 by LAP^2. Within the domain dom^3, a default route exists for forwarding packets to LAP^2. Moreover, packets can be forwarded to $\widetilde{LAP}^1$ and LAP^1 by using a default route within each of the domains $\widetilde{dom}^2$ and dom^2. Thus, dom^2 is a parent domain of dom^3, and dom^1 is a parent domain of all other domains. In this regard, $\widetilde{dom}^2$ is a sibling domain of dom^2 and dom^3, where dom^1 is their common parent domain. The locators $\widetilde{L}^1$ and L^1 describe the positions of $\widetilde{LAP}^1$ and LAP^1 within dom^1, and the locator L^2 describes the position of LAP^2 within dom^2. The endpoint S with the identifier ID_S is connected to an edge node in $\widetilde{dom}^2$. The position of S's edge node within $\widetilde{dom}^2$ is described by the locator L_S^2. The endpoint D with the identifier ID_D resides within dom^3, and the locator L_D^3 describes the position of D's edge node within dom^3.

In BPF-HiLLIS, the address of an endpoint consists a locator and its identifier. The address locator describes the position of the endpoint within the domain in which the endpoint is addressed. Table 5.1 gives the semi- and fully masked addresses of the endpoints S and D that are addressed within the four domains. Here, the masked locator of a LAP or an edge node is generated with its own public key by means of PEKS. Moreover, the identifiers of the endpoints are encrypted with their own public keys by means of PEKS.

	S's address		***D's address***	
	Semi-masked	***Fully masked***	***Semi-masked***	***Fully masked***
dom^1	$\widetilde{L}^1.E(ID_S)$	$E(\widetilde{L}^1).E(ID_S)$	$L^1.E(ID_D)$	$E(L^1).E(ID_D)$
$\widetilde{dom}^2$	$L_S^2.E(ID_S)$	$E(L_S^2).E(ID_S)$	$L^1.E(ID_D)$	$E(L^1).E(ID_D)$
dom^2	$\widetilde{L}^1.E(ID_S)$	$E(\widetilde{L}^1).E(ID_S)$	$L^2.E(ID_D)$	$E(L^2).E(ID_D)$
dom^3	$\widetilde{L}^1.E(ID_S)$	$E(\widetilde{L}^1).E(ID_S)$	$L_D^3.E(ID_D)$	$E(L_D^3).E(ID_D)$

Table 5.1: Semi- and fully masked addresses.

The endpoint S wants to send a packet to the endpoint D. We choose the masking rank "1" for the source address and the masking rank "2" for the destination address. For the bit-setting in the source and destination masking vectors, we begin with the most-significant bit. Thus, the masking vector for the source address is $(1, 0)$ which specifies full masking at level 1 and semi-masking at level 2. The endpoint S encrypts its own identifier with its public key assigned to the masking rank 1, and the destination identifier with D's public key assigned to the masking rank 2 (see Section 5.2.1). By means of the masked destination identifier, the endpoint S queries BPF-HiLLIS's mapping system for an appropriate destination address and for a destination masking vector whose blindness level is specified by the masking rank 2 (see Section 5.2.4). The endpoint S cannot itself generate the destination masking vector, since S does not know at which level the destination endpoint is located, and thus, the length of the destination masking vector. The mapping system responds with the masking vector $(1, 1, 0)$ and the address $E(L^1).E(ID_D)$. The destination masking vector specifies full masking at levels 1 and 2, and semi-masking at level 3. The endpoint S generates the packet

$$< 2 \mid 1 \mid (1, 1, 0) \mid (1, 0) \mid E(L^1).E(ID_D) \mid L_S^2.E(ID_S) >.$$

The LAPs on the route need the masking ranks for mapping lookups (see Section 5.2.2). In principle, the LAPs can determine the masking ranks by means of the masking vectors. For

that, the LAPs count the bits in the masking vectors and tries to discover the order of the bit-setting, which is a costly process. Therefore, the masking ranks are transferred in the packet. The packet is forwarded in the following steps (see Figure 5.2):

1. The packet is forwarded to $\widetilde{LAP}^1$ by using the default route.

2. $\widetilde{LAP}^1$ replaces the unmasked source locator L_S^2 with its masked locator $E(\widetilde{L}^1)$, since the source masking vector tells that the source address has to be fully masked at level 1. Thus, the LAP gets the packet

$$< 2 \mid 1 \mid (1,1,0) \mid (1,0) \mid E(L^1).E(ID_D) \mid E(\widetilde{L}^1).E(ID_S) >.$$

   The packet is forwarded to LAP^1 in the fully masked manner on the basis of the masked destination locator $E(L^1)$.

3. LAP^1 queries the mapping system for a fully masked address which is valid for the destination endpoint in dom^2. The mapping system responds with the address $E(L^2).E(ID_D)$. After replacing the destination address with the address from the mapping system, the LAP gets the packet

$$< 2 \mid 1 \mid (1,1,0) \mid (1,0) \mid E(L^2).E(ID_D) \mid E(\widetilde{L}^1).E(ID_S) >.$$

   On the basis of the new destination locator, the packet is forwarded to the next LAP in the fully masked manner.

4. Since the destination masking vector specifies semi-masking at level 3, LAP^2 queries the mapping system for a semi-masked address and gets the address $L_D^3.E(ID_D)$. The fully masked destination address is replaced with the semi-masked address from the mapping system, and the LAP gets the packet

$$< 2 \mid 1 \mid (1,1,0) \mid (1,0) \mid L_D^3.E(ID_D) \mid E(\widetilde{L}^1).E(ID_S) >.$$

   The packet is forwarded to the destination edge node in the semi-masked manner on the basis of the unmasked destination locator L_D^3. Eventually, the packet arrives at the destination edge node which resolves the masked destination identifier to a MAC address and forwards the packet to the MAC address.

5.2 Construction

BPF-HiLLIS adopts the level-based hierarchical scheme from BPF-HAIR and extends it so that endpoints can be attached to nodes at an arbitrary level. The top level of the hierarchy

is called *Core*. Each node in the Core belongs to one administrative domain and is under the control of a single ISP, just like in the today's Internet backbone. Nodes responsible for local networks are called *edge* nodes. For an endpoint connected to an edge node at level k, routing domains at levels 2 to k can be conceived as access providers or enterprise networks. The organisation of hierarchy levels is not restricted to be globally symmetrical. Domain providers can independently organise their own domains in various hierarchy levels.

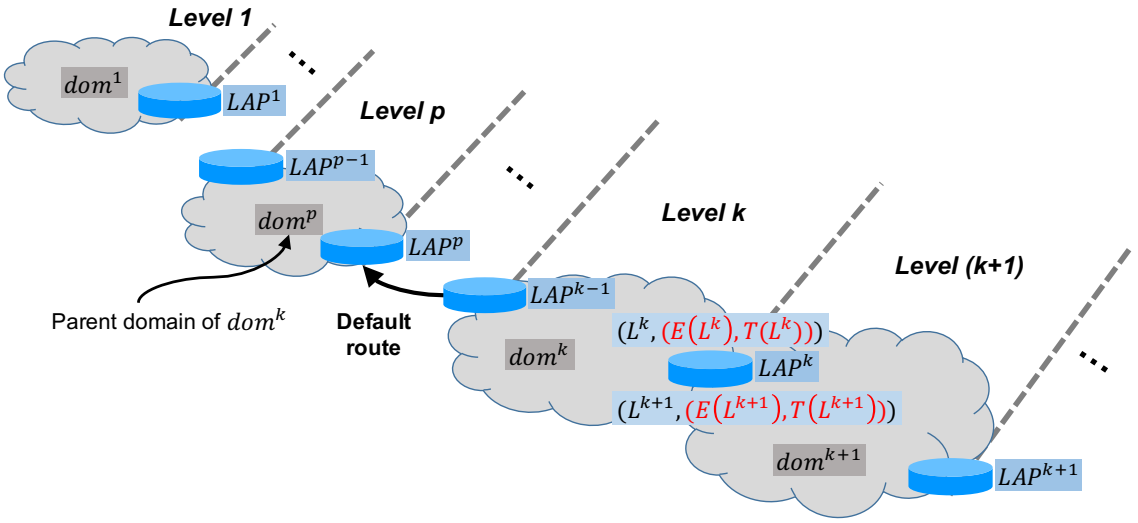


Figure 5.3: Level-based hierarchical scheme in BPF-HiLLIS.

A routing domain at level k is denoted by dom^k (see Figure 5.3). Each node in dom^k has a *k-level locator* L^k describing the level and location of the node in dom^k. Levels are connected via *Level Attachment Points (LAPs)* acting as gateway nodes. LAP^k denotes a LAP which connects dom^k and dom^{k+1} with each other. LAP^k has a k- and $(k+1)$-level locator L^k and L^{k+1}, which describe LAP's location in dom^k and dom^{k+1}. We call dom^p a *parent domain* of dom^k with $p < k$, if a default route via domains at levels $k-1$,...,$p+1$ exists for forwarding the packets from dom^k to dom^p.

For a node at level k, its *k-level masked locator and trapdoor* are $E(L^k)$ and $T(L^k)$ which are generated with its own public and private key by means of PEKS. Analogously, $E(ID_X)$ and $T(ID_X)$ denote the masked identifier and trapdoor of the endpoint X. We assume that all network nodes, communicating endpoints, and mapping systems have already generated a key pair using PEKS. Moreover, it is assumed that the public keys of the communicating endpoints are already exchanged and bound to their owners. Furthermore, we make the assumption that the communicating endpoints have already found the identifiers of each other, e.g., via DNS.

5.2.1 Addressing and masking

A *masking vector* mv_X^x for the endpoint X connected to an edge node in dom^x determines which masking mode has to be applied to its address at which level:

$$mv_X^x = (mt_X^1, ..., mt_X^x), \text{ where } mt_X^i = \begin{cases} 0 & \Rightarrow \text{Semi-masking in } dom^i \\ 1 & \Rightarrow \text{Full masking in } dom^i \end{cases}. \tag{5.1}$$

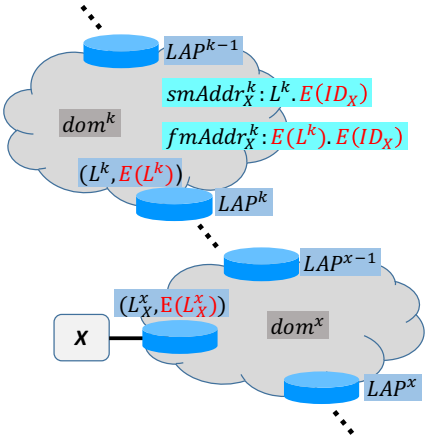
In dom^k, which is a child domain of dom^x with $x < k$, the address of the endpoint X is masked according to the masking type mt_X^x. The blindness level provided by a masking vector of the endpoint X is specified by its *masking rank* mr_X which is a natural number. In principle, up to 2^x masking ranks can be defined for the endpoint X. In Section 5.3.3, we define two blindness taxonomies into which certain masking ranks are classified.

It is decisive for the address of an endpoint, in which domain the endpoint has to be addressed (see Figure 5.4). For a given masking vector mv_X^x of the endpoint X connected to an edge node in dom^x, the *k-level (semi- or fully masked) address* of the endpoint is

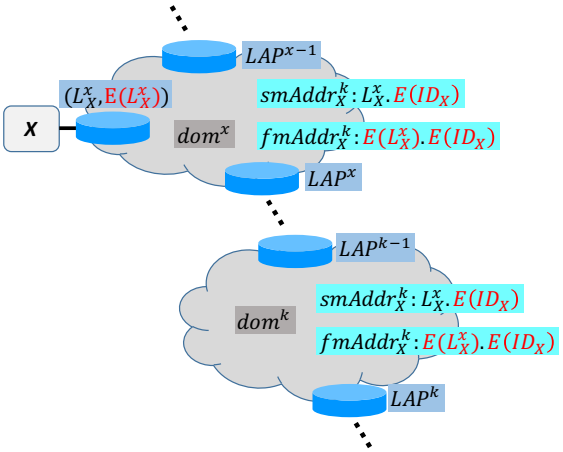
$$addr_X^k : \begin{cases} smAddr_X^k : Loc.E(ID_X) & \text{if } mt_X^k = 0 \\ fmAddr_X^k : E(Loc).E(ID_X) & \text{if } mt_X^k = 1 \end{cases}. \tag{5.2}$$

- ◇ If the endpoint X has to be addressed in a parent domain dom^k of dom^x with $1 \leq k < x$, $Loc = L^k$ and $E(Loc) = E(L^k)$ are k-level unmasked and masked locators of LAP^k via which dom^x can be reached (see Figure 5.4a).
- ◇ In dom^x and in a child domain dom^k of dom^x with $x < k$, $Loc = L_X^x$ and $E(Loc) = E(L_X^x)$ are the unmasked and masked locators of the edge node to which the endpoint X is connected (see Figure 5.4b).
- ◇ In case that dom^k is a sibling domain of dom^x, where dom^p is their next common parent domain, $Loc = L^p$ and $E(Loc) = E(L^p)$ are p-level unmasked and masked locators of LAP^p connecting dom^p with dom^{p+1} via which dom^x can be reached (see Figure 5.4c).

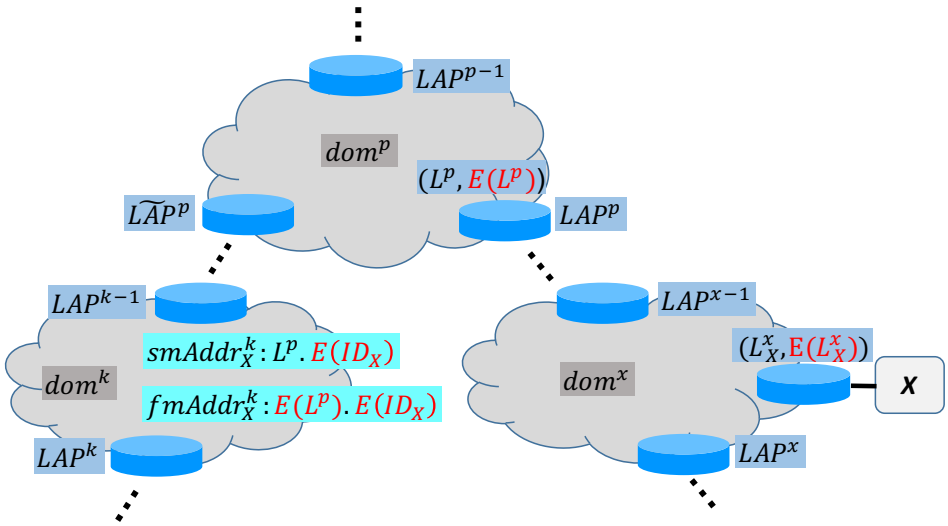
The masking vector associated with X's address determines level by level whether the address contains an unmasked or a masked locator. Moreover, the masking rank specifying the blindness level provided by the masking vector is assigned to the PEKS public key with which X's identifier has to be encrypted. For each masking rank, its own PEKS key pair is thus generated. The reason for that is discussed in Section 5.2.2.



(a) In a parent domain dom^k.



(b) In dom^x and in a child domain dom^k.



(c) In a sibling domain dom^k.

Figure 5.4: Addressing and masking in BPF-HiLLIS.

In dom^k, the header fields in a network packet from the source endpoint S in dom^s to the destination endpoint D in dom^d consist of

$$mr_D \mid mr_S \mid mv_D^d \mid mv_S^s \mid addr_D^k \mid addr_S^k \mid C(ID_S). \tag{5.3}$$

Both endpoints define their masking vectors according to equation 5.1, while the k-level addresses of the communicating endpoints are generated according to equation 5.2. More-

over, $C(ID_S)$ is the ciphertext generated by conventionally encrypting (e.g., with RSA) ID_S with the corresponding public key of the endpoint D. Here, $C(ID_S)$ serves the same purpose as in the previous BPF designs.

For packet generation, the source endpoint gets the cleartext identifier as well as the available masking ranks and the associated public keys of the destination endpoint, e.g., via DNS. The source endpoint chooses one of the masking ranks and encrypts the destination identifier with the associated public key. Moreover, the source endpoint selects one of its masking ranks as well as the associated masking vector already registered and encrypts its identifier with the public key assigned to the selected masking rank. The mapping and masking registration is discussed in detail below. The source endpoint gets the appropriate locator and masking vector of the destination endpoint by means of mapping and masking lookup also presented in detail below.

5.2.2 Mapping system

BPF-HiLLIS defines a hierarchical mapping system distributed at multiple levels (see Figure 5.5). MS^k denotes the mapping system in dom^k and is under the control of the domain provider. The mapping system at level 1 can be maintained by the providers of domains at level 2 in a distributed manner. MS^k maps k-level locators to the masked identifiers of endpoints connected to edge nodes at levels up to k, and maintains their masking ranks and vectors. The mappings are classified according to their masking ranks.

In MS^k with $k \leq x$, a *k-level (semi- or fully masked) mapping table entry* (MTE_X^k) for the endpoint X connected to an edge node in dom^x consists of a masking rank and vector as well as the masked identifier, trapdoor, and k-level unmasked or masked locator of the endpoint X:

$$MTE_X^k : \begin{cases} smMTE_X^k : [mr_X, mv_X^x, (E(ID_X), T(ID_X)), L^k] & \text{if } mt_X^k = 0 \\ fmMTE_X^k : [mr_X, mv_X^x, (E(ID_X), T(ID_X)), E(L^k)] & \text{if } mt_X^k = 1 \end{cases}.$$

- ◇ In case of $k < x$, L^k and $E(L^k)$ are the k-level unmasked and masked locators of LAP^k connecting dom^k with dom^{k+1} via which dom^x can be reached.
- ◇ If $k = x$, both locators are the x-level unmasked and masked locators L_X^x and $E(L_X^x)$ of the edge node to which the endpoint X is connected.

In contrast to the encryption function of PEKS, its trapdoor function is deterministic. This means that the function always outputs the same trapdoor value for a pair of a cleartext

and a private key (see Section 2.1). If MS^k maintains X's semi- and fully masked mappings whose trapdoors are generated with the same private key, their byte values are the same, and thus, MS^k can find out that both mappings belong to the same endpoint. In this way, MS^k can disclose that $E(L^k)$ signifies L^k. Thus, the mapping system can make void the effect of the full blindness (see Section 5.3.2.2). With regard to the discussion above, trapdoors in mappings for different masking ranks belonging to the endpoint X have to be generated with different private keys in order to maintain multiple mappings for X without making void the effect of the full blindness. Hence, the endpoint X generates a new key pair for each masking rank. To register a mapping for the masking rank mr_X, the endpoint uses its masked identifier and trapdoor which are generated with the public/private key assigned to the masking rank (see Section 5.2.3). In this way, the byte values of trapdoors in the mapping entries for the endpoint X differ from each other.

$Test(E(ID_X), T(ID_X))$ outputs 1, if and only if $E(ID_X)$ and $T(ID_X)$ are generated with the same key pair (X_{pub}, X_{priv}). Therefore, a mapping lookup request for the masking rank mr_X contains X's masked identifier which is generated with X's public key assigned to the masking rank mr_X (see Section 5.2.4). Moreover, upon receiving an mapping lookup request for the masking rank mr_X, the mapping system performs $Test()$ with the masked identifier from the request and the trapdoor in each mapping entry with the same masking rank as parameters. Thus, the mapping system tries to find an entry containing the trapdoor which is generated with X's private key assigned to the masking rank mr_X, and $Test()$ outputs 1, if such an entry exists. Due to that reason, network packets have to contain the masking ranks for the source and destination endpoints, and the source and destination masked identifiers have to be generated with the public keys assigned to the source and destination masking ranks.

Except the mapping system at level 1, MS^k keeps the $(k-1)$-level unmasked locators and the PEKS key pairs of LAPs connecting dom^k with dom^{k-1}. Since these LAPs are under the control of the domain provider which also services the mapping system, an additional system is not needed to certify the keys. Moreover, MS^k holds the semi- and fully masked addresses of MS^{k-1} in dom^{k-1} which is a parent domain of dom^k, if the mapping/masking registration and lookup traffic has to be masked. These addresses can be obtained from the provider of dom^{k-1}.

An enhanced DHCP server in a local network keeps the unmasked locator, the PEKS key pair, and the level of the edge node responsible for the local network. Moreover, the DHCP server holds the semi- and fully masked addresses of the mapping system responsible for the domain to which the local network belongs. The local network provider can get these addresses from

the domain provider and configure the DHCP server with them. If the destinations of the mapping/masking registration and lookup messages do not have to be masked, the DHCP server maintains only the cleartext address of the mapping system.

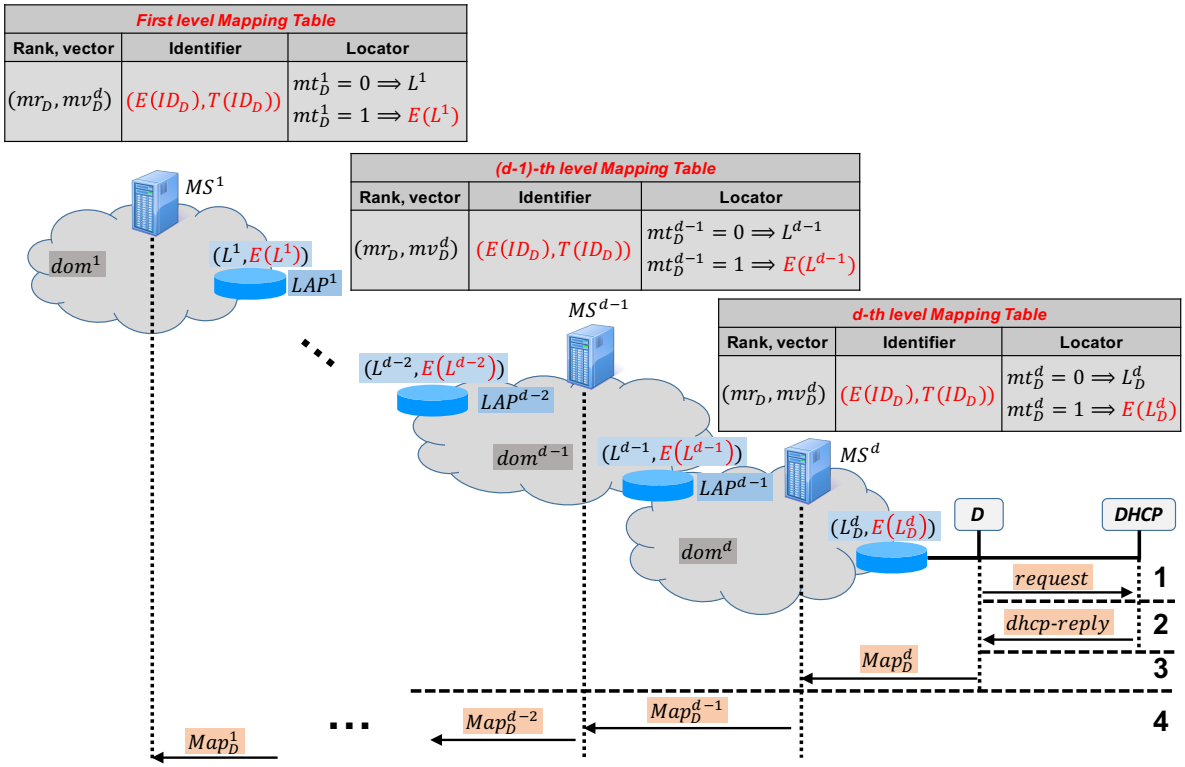


Figure 5.5: Mapping and masking registration in BPF-HiLLIS.

5.2.3 Mapping and masking registration

A new endpoint in a local network has to register the mapping of its identifier to its actual locator at the mapping system. By means of the registration, the endpoint also defines in which masking mode at which level a packet addressed to itself has to be forwarded. For mapping and masking registration of the endpoint D newly connected to a local network in dom^d, it is proceeded as follows (see Figure 5.5):

1. The endpoint D broadcasts a request in its local network.

2. The DHCP server in the local network responds with the trapdoor, d-level unmasked and masked locators of the edge node responsible for the local network. Moreover, the reply message contains the d-level semi- and fully masked addresses of the mapping system MS^d which is responsible for dom^d:

$$dhcp\text{-}reply\{L_D^d, E(L_D^d), T(L_D^d), smAddr_{MS^d}^d, fmAddr_{MS^d}^d\}.$$

3. The endpoint defines a masking rank and vector (mr_D, mv_D^d) and generates a new key pair by means of PEKS. After that, the endpoint encrypts its identifier with the new public key and generates the trapdoor for its identifier with the new private key by means of PEKS. Thus, the endpoint gets the tuple $(E(ID_D), T(ID_D))$. Finally, the endpoint sends its *d-level (semi- or fully masked) mapping* to MS^d:

$$Map_D^d : \begin{cases} smMap_D^d : \{mr_D, mv_D^d, (E(ID_D), T(ID_D)) \Rightarrow L_D^d\} & \text{if } mt_D^d = 0 \\ fmMap_D^d : \{mr_D, mv_D^d, (E(ID_D), T(ID_D)) \Rightarrow E(L_D^d)\} & \text{if } mt_D^d = 1 \end{cases}.$$

   In case of semi-masked mapping, the endpoint uses its own semi-masked address newly configured and the semi-masked address of the mapping system. Otherwise, the destination and source addresses of the mapping message are the fully masked addresses of the mapping system and endpoint.

4. For the incoming k-level mapping of the endpoint D with $k \leq d$, MS^k performs

$$Test(E(ID_i), T(ID_D))$$

   for each mapping table entry i which contains the same masking rank as mr_D.

   - ◇ If $Test()$ returns 1 for an entry, i.e. an entry for the endpoint D already exists, the entry is updated with the locator and masked identifier from the incoming mapping message.
   - ◇ If such an masking rank is not registered so far, or $Test()$ returns 0 for all entries with the same masking rank, a new entry is created with the values from the message. Moreover, MS^k sends the $(k-1)$-level mapping of the endpoint D to MS^{k-1}:

$$Map_D^{k-1} : \begin{cases} smMap_D^{k-1} : \{mr_D, mv_D^d, (E(ID_D), T(ID_D)) \Rightarrow L^{k-1}\} & \text{if } mt_D^{k-1} = 0 \\ fmMap_D^{k-1} : \{mr_D, mv_D^d, (E(ID_D), T(ID_D)) \Rightarrow E(L^{k-1})\} & \text{if } mt_D^{k-1} = 1 \end{cases}.$$

     Here, L^{k-1} and $E(L^{k-1})$ are the $(k-1)$-level unmasked and masked locators of one of the LAPs connecting dom^k with dom^{k-1} which is the parent domain of dom^k. As in step 3, the semi- or fully masked addresses of MS^{k-1} and MS^k are used as the destination and source addresses of the mapping message in case of semi- or fully masked mapping.

   This process is performed analogously up to MS^1.

By repeating the steps 3 and 4 with another masking rank and vector tuples, the endpoint D can register multiple masking ranks and vectors. The masking vector $(0, .., 0)$ is registered by default. This masking vector corresponds to the lowest masking rank. By means of an additional parameter in the registration message, the endpoint D can specify up to which level a masking rank has to be registered. If the mapping registration message arrives at a mapping system at the specified level, the message is not forwarded to the mapping system in the parent domain anymore. Thus, only endpoints at up to the specified level can communicate with the endpoint D according to the registered masking rank. This feature is needed for the approach in Section 5.2.8.

If the mapping registration traffic does not have to be masked, the endpoint D gets the cleartext address of MS^d and uses it as the destination address of the edge mapping message. Moreover, MS^k can use its own cleartext address and the unmasked address of MS^{k-1} as the source and destination addresses of mapping messages transferred between them.

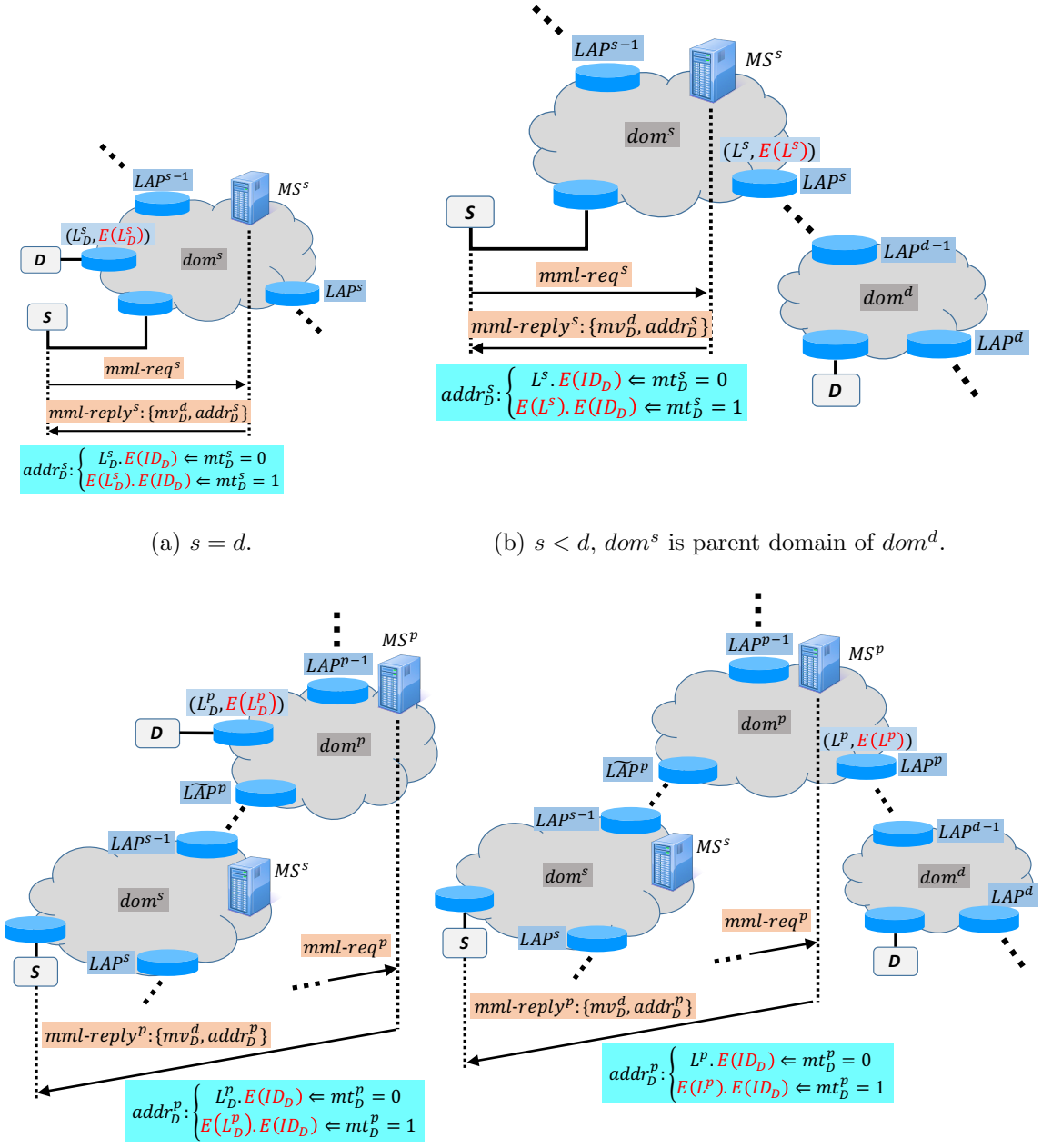
The endpoint D periodically sends its mappings serving as keep-alive messages. If a keep-alive message is missing, e.g., because of moving to another domain, the mapping entries for D are deleted by the mapping system. Due to change of security properties in domains, the endpoint can intend to delete a masking rank and vector tuple (mr_D, mv_D^d) already registered. For that, the endpoint sends a delete message

$$Delete\_Map_D : \{mr_D, mv_D^d, T(ID_D)\} \text{ to } MS^d.$$

Here, $T(ID_D)$ is the trapdoor created with the private key which the endpoint D has generated for the registration of this masking rank and vector tuple. For the incoming masking delete message with $k \leq d$, MS^k determines the entry holding the mapping and masking information for the endpoint D by performing $Test()$. Here, the trapdoor from the delete message and the masked identifier in each entry with the same masking rank as mr_D are taken as parameters. After finding the entry, MS^k deletes the entry and forwards the delete message to the mapping service in dom^{k-1} which is the parent domain of dom^k. This proceeding is repeated up to MS^1 analogously.

5.2.4 Mapping and masking lookup

To send a packet from the source endpoint S in dom^s to the destination endpoint D in dom^d, the source endpoint chooses a masking rank specifying the blindness level to be applied for the destination address. After encrypting the destination identifier with D's public key assigned to the selected masking rank mr_D, the source endpoint has to find an appropriate locator of the destination endpoint and the masking vector providing the selected blindness level. For that, it is proceeded as follows (see Figure 5.6):



(a) $s = d$.

(b) $s < d$, dom^s is parent domain of dom^d.

(c) $d = p$, dom^d is parent domain of dom^s.

(d) dom^d is sibling domain of dom^s.

Figure 5.6: Mapping and masking lookup in BPF-HiLLIS.

1. The source endpoint sends a *s-level mapping and masking lookup request* to the mapping system MS^s in dom^s:

$$mml\text{-}req^s : \{mr_D, E(ID_D), mr_S, mv_S^s, addr_S^s\}.$$

Here, $addr_S^s$ is the s-level address of the requesting endpoint S. Moreover, the masking rank mr_S specifies the blindness level provided by the masking vector mv_S^s which is selected to mask the source address of the packets for the communication with the endpoint D. These values are required to address the requesting endpoint and to accordingly mask its address in a reply message.

2. For the incoming request, MS^s performs

$$Test(E(ID_D), T(ID_i))$$

for each mapping table entry i with the same masking rank as mr_D.

   - ◇ If $Test()$ returns 1 for an entry, it means that either $s = d$, i.e. both endpoints are connected to edge nodes in the same domain (see Figure 5.6a) , or dom^s is a parent domain of dom^d with $s < d$ (see Figure 5.6b). Thus, MS^s sends the *s-level mapping and masking lookup reply* back to the endpoint S:

$$mml\text{-}reply^s : \{mv_D^d, addr_D^s\}$$

     - ◇ In case of $s = d$, $addr_D^s$ contains the unmasked or masked locator of the edge node to which D is connected.
     - ◇ Otherwise (i.e, $s < d$), the address in the reply message contains the s-level unmasked or masked locator of LAP^s via which dom^d can be reached.

   - ◇ If $Test()$ returns 0 for all entries, the locator in the address of the requesting endpoint is updated with the $(s-1)$-level locator of LAP^{s-1} connecting dom^s with dom^{s-1} which is the next parent domain of dom^s. Here, mt_S^{s-1} determines whether the $(s-1)$-level unmasked or masked locator of LAP^{s-1} is taken for updating the locator. Thus, the s-level address of S is updated with its $(s-1)$-level address. After that, the $(s-1)$-level mapping and masking lookup request mml_req^{s-1} is forwarded to the mapping system in the next parent domain.

3. For the incoming p-level mapping and masking lookup request mml_req^p with $p < s$, the mapping system MS^p in dom^p, which is a parent domain of dom^s, performs $Test()$ analogously in oder to find an entry holding the mapping and masking information for the requested masking rank and masked identifier.

   - ◇ If $Test()$ returns 1 for an entry, it means that either $d = p$ and dom^d is a parent domain of dom^s (see Figure 5.6c), or dom^d is a sibling domain of dom^s and dom^p is the next common parent domain of dom^s and dom^d (see Figure 5.6d). Thus, MS^p responds with D's masking vector and p-level address $addr_D^p$.

- ◇ In case of $d = p$, $addr_D^p$ contains the unmasked or masked edge locator of the endpoint D according to mt_D^p.
- ◇ If the endpoint D resides in a sibling domain of dom^s, the locator of $addr_D^p$ is the p-level unmasked or masked locator of LAP^p via which dom^d can be reached.

◇ If $Test()$ returns 0 for all entries in MS^p, the address of the requesting endpoint is updated, and the request is forwarded to the mapping system in the next parent domain in the same manner as described above.

4. Step 3 is analogously performed until $Test()$ returns 1 in a parent domain of dom^s. If $Test()$ also returns 0 for all entries in MS^1, the requesting endpoint S gets an "unregistered" message.

If the endpoint S gets an unregistered message, it tries to perform a new lookup with another masking rank or with the lowest masking rank which is registered by default. The endpoint S can cache the values from the reply message so that it does not have to perform a lookup for each packet. Moreover, the endpoint S can use the unmasked address of MS^s, if the mapping lookup traffic does not have to be masked. Furthermore, MS^k can also use its own unmasked address and the unmasked address of MS^{k-1} as the source and destination addresses of an forwarded request message.

For a packet, which comes from the domain at the upper level and contains L^k or $E(L^k)$ as its k-level destination locator, LAP^k queries MS^{k+1} for the $(k+1)$-level address of the destination endpoint D. The request message contains the masked destination identifier as well as the destination masking rank and vector of the packet. For the lookup request, MS^{k+1} performs $Test()$ with the masked identifier from the request message and trapdoor in each mapping table entry containing the requested masking rank as parameters. Thus, the mapping system determines the entry holding the mapping and masking information for the endpoint D and responds with D's $(k+1)$-level address $addr_D^{k+1}$. LAP^k can cache the locator from the reply message for further packets addressed to the endpoint D.

5.2.5 Enhanced masked routing

If both the unmasked and masked routing in Section 4.1.2.1 are performed in a domain, the network node M has the following unmasked and masked routing table entries for the network node N with the locator Loc_N:

$$umRTE_N : [Loc_N, P_N, D_N] \text{ and } mRTE_N : [(E(Loc_N), T(Loc_N)), P_N, D_N]$$

Here, P_N is the number of the port via which the network node N can be reached, and D_N is the distance to the network node N. Moreover, $E(Loc_N)$ and $T(Loc_N)$ are N's masked

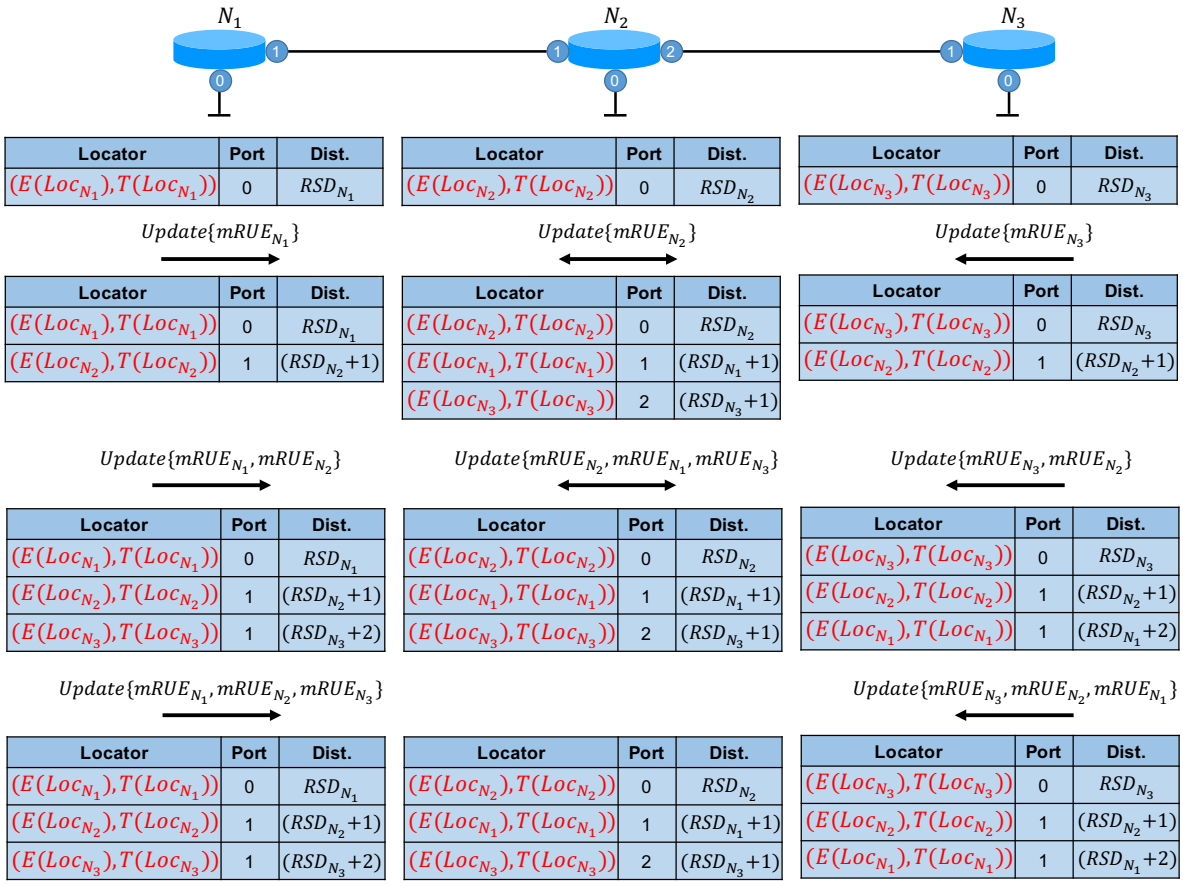


Figure 5.7: Enhanced masked routing in BPF-HiLLIS.

locator and trapdoor which are generated with its public and private key by means of PEKS.

For an incoming fully masked packet containing $E(Loc_N)$ as its destination locator, the network node M performs $Test()$ with the masked destination locator and the trapdoor in each masked routing table entry as parameters in order to determine the port via which the packet has to be forwarded. Thus, $Test()$ returns 1 for the entry $mRTE_N$. But the network node M can compare the tuple (P_N, D_N) from $mRTE_N$ with the tuple (P_i, D_i) in each unmasked routing table entry $umRTE_i$. Eventually, the tuples in $mRTE_N$ and $umRTE_N$ match each other. Thus, the network node M finds out that $E(Loc_N)$ signifies Loc_N. This means that the network node discloses the destination locator of the incoming packet. In this way, network nodes can make void the effect of the full blindness by exploiting their unmasked routing tables.

In order to resolve this issue, we enhance the masked routing (see Figure 5.7). Here, a

masked routing table entry for the network node N is

$$mRTE_N : [(E(Loc_N), T(Loc_N)), P_N, \underbrace{(RSD_N + D_N)}_{RD_N}].$$

Here, RD_N is a random distance to the network node N, which is the sum of N's random start distance RSD_N and real distance D_N. For the cold start in the enhanced masked routing, the network node N puts the random start distance $RSD_N \geq 2$ (instead of 0) as the distance to its local network into the first masked routing table entry. A masked routing update entry for the network node N contains its masked locator, trapdoor, and random distance:

$$mRUE_N : [(E(Loc_N), T(Loc_N)), RD_N].$$

For the masked routing update, network nodes proceed in the same manner as described in Section 3.1.3. Figure 5.7 visualises the enhanced masked routing for a basic topology consisting of three network nodes. Thus, a network node cannot state with certainty that the masked locator in a masked routing table entry signifies the cleartext locator in an unmasked routing table entry, if the port-distance tuples in both entries match each other.

The unmasked and masked routing is performed domain-wise as in BPF-HAIR. Thus, a network node in dom^k keeps unmasked and masked routing information only for the nodes in dom^k. Moreover, LAP^k maintains two unmasked and masked routing tables for dom^k and dom^{k+1}. Furthermore, each node in dom^k holds an unmasked and a masked routing table entry defining the default route to LAP^{k-1} connecting dom^k with dom^{k-1}.

5.2.6 Selective masked routing table entry setup

Masked routing table setup in an entire domain is a costly process as evaluated in Sections 4.1.7 and 4.2.6. In principle, only network nodes on the route between two communicating endpoints need to maintain according masked routing table entries to enable the fully blind packet forwarding for these two endpoints. While unmasked routing is performed by default, we propose multiple approaches for different cases to selectively set up masked routing table entries.

5.2.6.1 Case 1

If a masked routing table entry for a single network node A has to be set up at each node in a domain, it is proceeded as follows (see Figure 5.8):

1. The network node A encrypts its locator with its public key and generates the trapdoor for the locator with its private key by means of PEKS. After that, A chooses a random start distance RSD_A as the distance to its local network and creates a masked routing

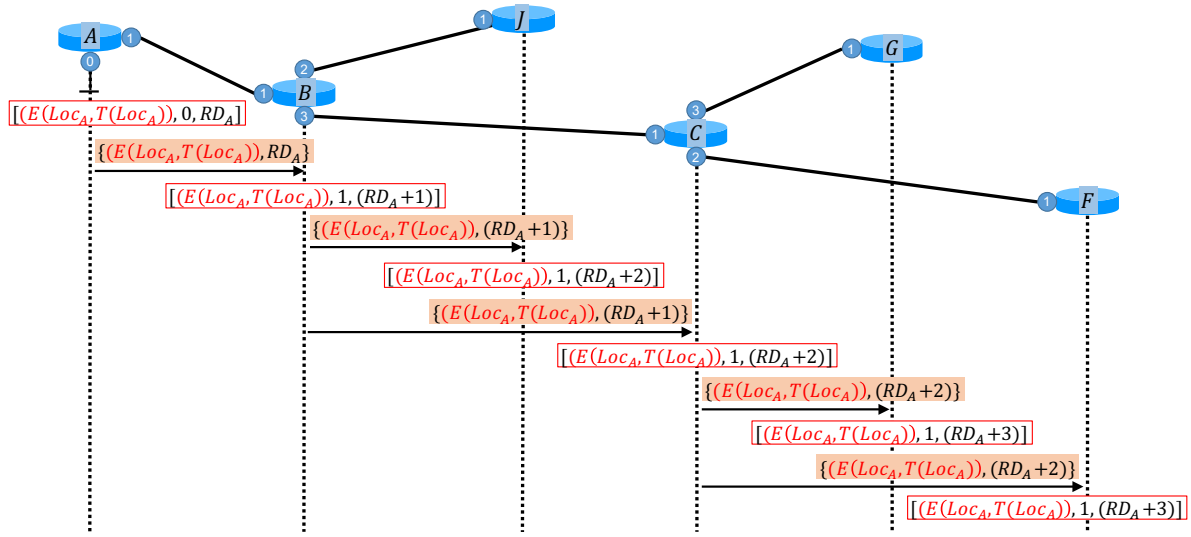


Figure 5.8: Selective masked routing table entry setup in case 1.

table entry with its masked locator, trapdoor, random distance, and port number for its local network. Finally, A broadcasts a *undirected masked routing table entry setup (undrct-mRTE-set)* message for itself:

$$undrct\text{-}mRTE\text{-}set_A : \{(E(Loc_A), T(Loc_A)), RD_A\}.$$

2. For the incoming setup message, each network performs

$$Test(E(Loc_A), T(Loc_i))$$

for each masked routing table entry i.

   - ◇ If $Test()$ returns 1 for an entry i, it means that an entry is already set up for the network node A. In this case, the distances in the entry and setup message are compared with each other:
     - ◇ If the distance[1] in the entry is greater than $(RD_A + 1)$, the entry is updated with the masked locator $E(Loc_A)$, the distance $(RD_A + 1)$, and the number of the port via which the setup message has been received.
     - ◇ Otherwise, only the masked locator in the entry is updated with the masked locator from the setup message.
   - ◇ If $Test()$ returns 0 for all entries, a new entry is created with $E(Loc_A)$, $T(Loc_A)$, $(RD_A + 1)$, and the number of the port via which the network node has received the setup message.

[1] Hop count

If a new entry is created, or the distance in an entry is updated, the network node sends the updated setup message

$$undrct\text{-}mRTE\text{-}set_A : \{(E(Loc_A), T(Loc_A)), (RD_A + 1)\}$$

to all of the neighbours except the neighbour that the setup message has arrived from.

After the setup has terminated, endpoints can send packets to the endpoints connected to the network node A in the fully masked manner.

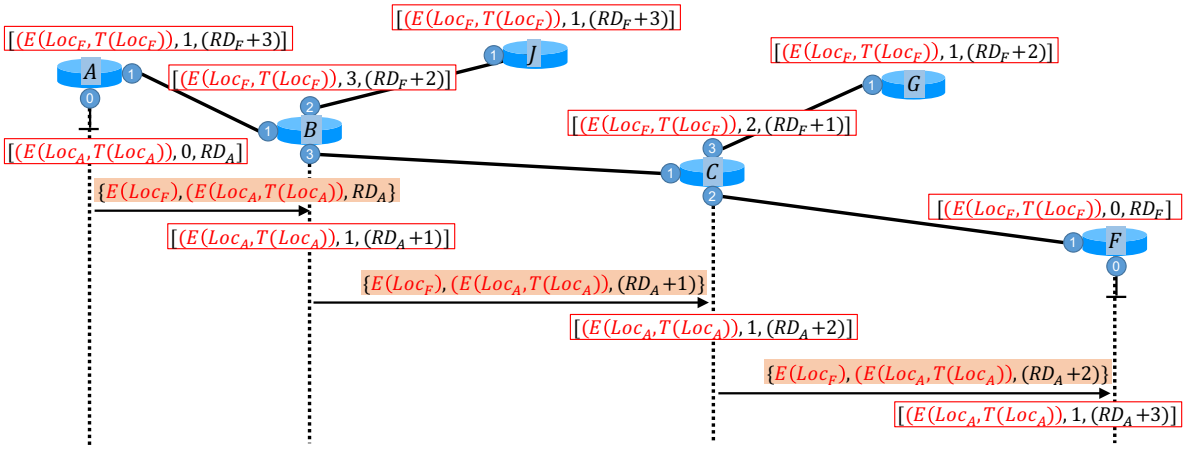


Figure 5.9: Selective masked routing table entry setup in case 2.

5.2.6.2 Case 2

If a network node A intends to set up masked routing table entries for itself only at the nodes on the route to a selected network node F for which masked routing table entries are already set up at each node, it is proceeded as follows (see Figure 5.9):

1. The network node A first creates a masked routing table entry for itself as described in step 1 in Section 5.2.6.1. Next, A performs $Test()$ with F's masked locator and the trapdoor in each masked routing table entry as parameters. Finally, A sends a *directed masked routing table entry setup (drct-mRTE-set)* message via the port mapped in the entry for which $Test()$ returns 1:

$$drct\text{-}mRTE\text{-}set_{A \to F} : \{E(Loc_F), (E(Loc_A), T(Loc_A)), RD_A\}.$$

2. For the incoming setup message, a network node N performs

$$Test(E(Loc_F), T(Loc_N))$$

in order to determine whether it is the target node of the setup message.

- ◇ If $Test()$ returns 1, it means that N is addressed as the network node at which the setup has to terminate. In this case, N creates a new entry or updates an already existing one for the network node A as proposed in step 2 in Section 5.2.6.1.
- ◇ Otherwise, the network node proceeds in the same manner in order to create a new entry or to update an already existing one for the network node A. After that, N performs $Test()$ with F's masked locator and the trapdoor in each masked routing table entry as parameters. Finally, the updated setup message is sent via the port mapped in the entry for which $Test()$ returns 1:

$$drct\text{-}mRTE\text{-}set_{A \rightarrow F} : \{E(Loc_F), (E(Loc_A), T(Loc_A)), (RD_A + 1)\}.$$

In this way, a *blind route* is set up between the network nodes A and F so that the endpoints connected to both network nodes can communicate with each other in the fully blind mode.

5.2.6.3 Case 3

Here, it is the same case as in Section 5.2.6.2, but no masked routing table entries are set up for the selected network node F. In this case, a unidirectional blind route is first set up, and after that, the route is made bidirectional. For that, we propose two approaches:

Approach 1

1. After creating a masked routing table entry for itself, the network node A broadcast a directed setup message containing a time-to-live value (see Figure 5.10):

$$drct\text{-}mRTE\text{-}set_{A \rightarrow F} : \{E(Loc_F), (E(Loc_A), T(Loc_A)), RD_A, TTL\}.$$

2. For the incoming setup message, a network node N checks whether it is addressed in the message by proceeding in the same manner as introduced in step 2 in Section 5.2.6.2.
   - ◇ If this is the case, N creates a new entry for the network node A and goes on to step 3.
   - ◇ Otherwise, N creates a temporary entry for A including the time-to-live value and starts a countdown timer for the temporary entry. After that, the updated setup message is sent via all ports except the port via which the setup message has arrived:

$$drct\text{-}mRTE\text{-}set_{A \rightarrow F} : \{E(Loc_F), (E(Loc_A), T(Loc_A)), (RD_A + 1), TTL\}.$$

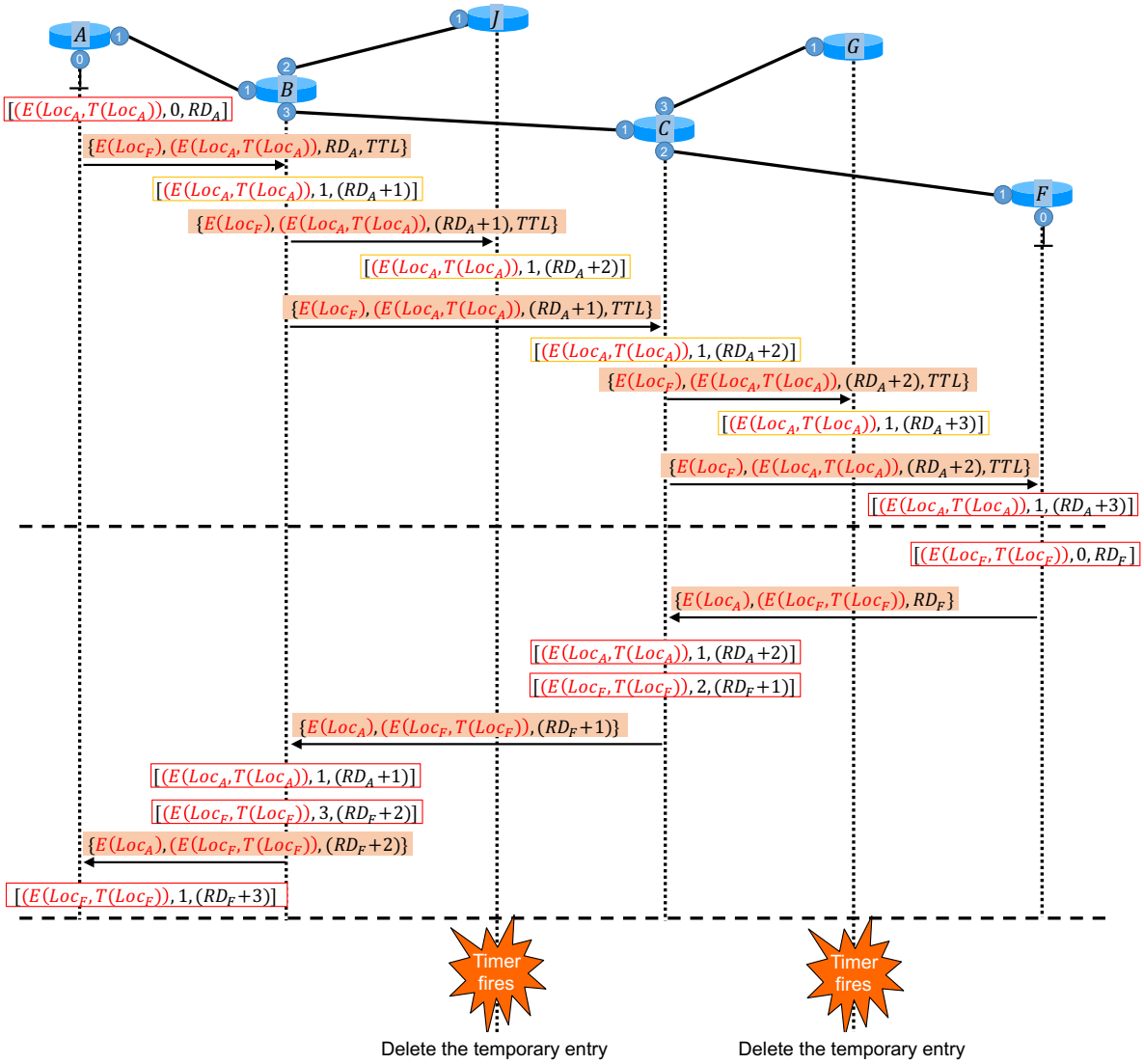


Figure 5.10: Selective masked routing table entry setup by means of approach 1 in case 3.

3. If the setup message arrives at the network node F, which is specified in the message, the network node F creates entries for A and itself. Thus, a unidirectional blind route is set up between the network nodes A and F. Now, the route has to be made bidirectional. For that, the network node F sends a directed setup message for itself via the port specified in the entry for the network node A:

$$drct\text{-}mRTE\text{-}set_{F\rightarrow A} : \{E(Loc_A), (E(Loc_F), T(Loc_F)), RD_F\}.$$

4. For the incoming setup message, a network node N performs $Test()$ with A's masked locator and its own trapdoor as parameters.

◇ If $Test()$ returns 0, N performs $Test()$ with A's masked locator and the trapdoor in each temporary masked routing table entry as parameters. The entry for which $Test()$ returns 1 is made permanent. After that, N creates a new entry or updates an already existing one for the network node F as introduced in step 2 in Section 5.2.6.1. Eventually, the updated setup message is sent via the port specified in the entry which is made permanent:

$$drct\text{-}mRTE\text{-}set_{F \rightarrow A} : \{E(Loc_A), (E(Loc_F), T(Loc_F)), (RD_F + 1)\}.$$

If $Test()$ returns 0 for all temporary entries, the setup message is dropped.

◇ Otherwise (i.e., the message has arrived at the network node A), a new masked routing table entry for F is created, and the setup terminates.

5. If the countdown timer for a temporary entry fires at a node, the node deletes the entry.

Approach 2

The flooding in approach 1 can be avoided by revealing only the one end-node of the blind route between the network nodes A and F. Here, we leverage the unmasked routing tables entries which are set up by default (see Figure 5.11):

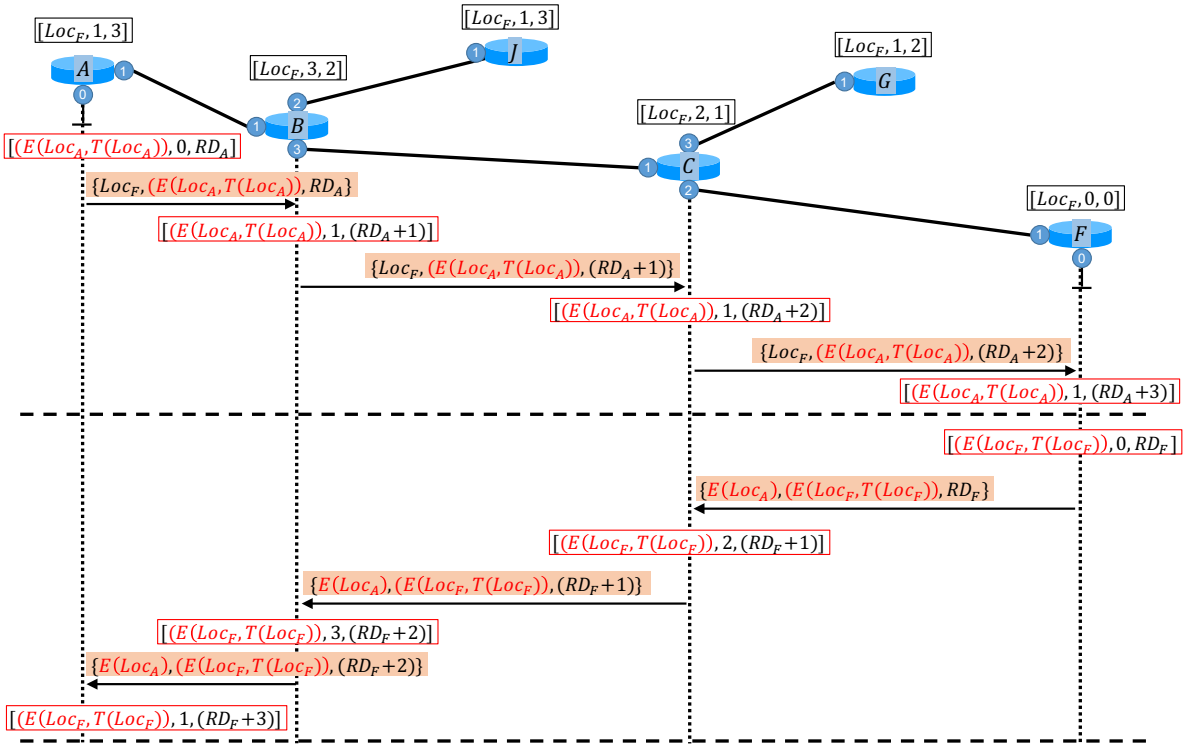


Figure 5.11: Selective masked routing table entry setup by means of approach 2 in case 3.

1. The network node A creates a masked routing table entry for itself. After that, A sends a directed setup message containing F's unmasked locator via the port mapped in the unmasked routing table entry for F:

$$drct\text{-}mRTE\text{-}set_{A \to F} : \{Loc_F, (E(Loc_A), T(Loc_A)), RD_A\}.$$

2. For the incoming setup message, a network node N checks whether it is specified as the network node at which the setup has to terminate.

   ◇ If this is the case, it means that the setup message has arrived at its destination node F. In this case, the network node creates masked routing table entries for A and itself and continues with step 3.

   ◇ Otherwise, the network node creates a new entry or updates an already existing one for A. After that, the updated setup message is sent via the port mapped in the unmasked routing table entry for F:

$$drct\text{-}mRTE\text{-}set_{A \to F} : \{Loc_F, (E(Loc_A), T(Loc_A)), (RD_A + 1)\}.$$

3. After creating masked routing table entries for A and itself, the network node F sends a directed setup message for itself via the port which is specified in the entry for A:

$$drct\text{-}mRTE\text{-}set_{F \to A} : \{E(Loc_A), (E(Loc_F), T(Loc_F)), RD_F\}.$$

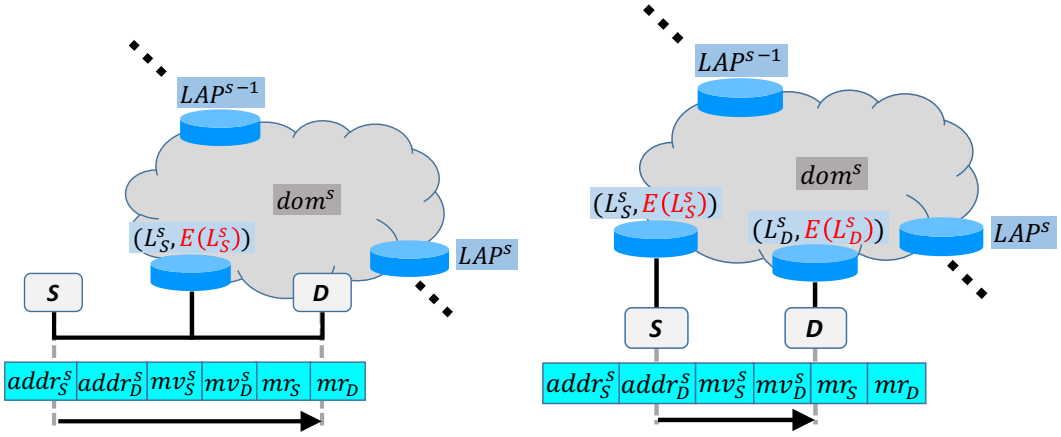
4. For the incoming setup message, network nodes on the route to the network node A proceed analogously in order to create entries for the network node F.

5.2.7 Packet delivery

The source endpoint S in dom^s wants to send a packet to the destination endpoint D in dom^d. Here, the source and destination addresses of the packet have to be masked according to the selected masking ranks mr_S and mr_D. It is assumed that both endpoints have already registered their masking ranks and the associated masking vectors as proposed in Section 5.2.3. By means of PEKS, the source endpoint first encrypts the source and destination identifiers with its own and D's public keys which are assigned to the selected masking ranks. After that, the source endpoint finds D's appropriate locator and masking vector as introduced in Section 5.2.4. Based on the locations of the domains dom^s and dom^d relative to each other, we have the following cases.

5.2.7.1 Intra-domain packet forwarding

If the source and destination endpoints S and D reside in the same domain (see Figure 5.12), i.e. $s = d$, the source endpoint gets D's masking vector mv_D^s and s-level address $addr_D^s$ containing D's unmasked or masked edge locator, and it thus generates the packet



(a) Case 1. (b) Case 2.

Figure 5.12: Intra-domain packet forwarding in BPF-HiLLIS.

$$< mr_D \mid mr_S \mid mv^s_D \mid mv^s_S \mid addr^s_D \mid addr^s_S >.$$

The source endpoint checks the masking type mt^s_D in D's masking vector mv^s_D.

- ◇ If $mt^s_D = 0$, D's address is semi-masked. In this case, the source endpoint checks whether the source and destination addresses have the same unmasked locator.
  - ◇ If $L^s_S = L^s_D$, i.e. both endpoints are connected to the same edge node, the source endpoints resolves the masked destination identifier to the MAC address of the destination endpoint and sends the packet to this MAC address (see Figure 5.12a).
  - ◇ Otherwise, the packet is sent to the edge node to which the source endpoint is connected (see Figure 5.12b).
- ◇ If $mt^s_D = 1$, the destination address contains D's masked edge locator $E(L^s_D)$. In this case, S performs $Test()$ with the masked destination locator and the trapdoor for S's edge locator as parameters.
  - ◇ If $Test()$ returns 1, i.e. the source and destination endpoints are connected to the same local network, the masked destination identifier is resolved to D's MAC address, and the packet is sent to the destination endpoint (see Figure 5.12a).
  - ◇ Otherwise, the source endpoint sends the packet to its edge node (see Figure 5.12b).

For the incoming packet, a network node in dom^s checks whether mt^s_D is set.

- ◇ If this is the case, i.e. the destination address is fully masked, the network node performs $Test()$ with the masked destination locator and the trapdoor in each masked routing table entry as parameters. The packet is then forwarded via the port mapped in the entry for which $Test()$ returns 1.
- ◇ Otherwise, the network node performs conventional longest prefix matching and sends the packet via the port mapped in the matched entry.

Eventually, the packet arrives at the edge node to which the destination endpoint is connected. The destination edge node determines that the packet is addressed to its own local network. After resolving the masked destination identifier to D's MAC address, the packet is sent to this MAC address.

5.2.7.2 Top-down packet forwarding

If dom^s is a parent domain of dom^d with $s < d$, the source endpoint gets D's masking vector mv_D^d and s-level address $addr_D^s$ which contains the s-level unmasked or masked locator of LAP^s via which dom^d can be reached. Thus, the source endpoint creates the packet

$$< mr_D \mid mr_S \mid mv_D^d \mid mv_S^s \mid addr_D^s \mid addr_S^s >.$$

To send the packet to the destination endpoint, the following steps are performed (see Figure 5.13):

1. According to the masking type mt_D^s, the packet is forwarded to LAP^s in semi- or fully masked manner.
2. For the incoming packet with $s \leq k < d-1$, LAP^k proceeds as follows:
   - ◇ If $mt_D^{k+1} = 0$, i.e. the destination address has to be semi-masked in dom^{k+1}, LAP^k gets D's $(k+1)$-level address $addr_D^{k+1}$ from MS^{k+1} in dom^{k+1}. This address contains the $(k+1)$-level unmasked locator of LAP^{k+1} via which to reach dom^d.
   - ◇ If the masking type mt_D^{k+1} is set, LAP^k queries MS^{k+1} for D's $(k+1)$-level address $addr_D^{k+1}$ which contains the $(k+1)$-level masked locator of LAP^{k+1}.

   After that, LAP^k replaces the k-level destination address of the packet with $addr_D^{k+1}$ and gets the packet

   $$< mr_D \mid mr_S \mid mv_D^d \mid mv_S^s \mid addr_D^{k+1} \mid addr_S^s >.$$

   According to the masking type mt_D^{k+1}, the packet is forwarded to LAP^{k+1} in the semi- or fully masked manner within dom^{k+1}.

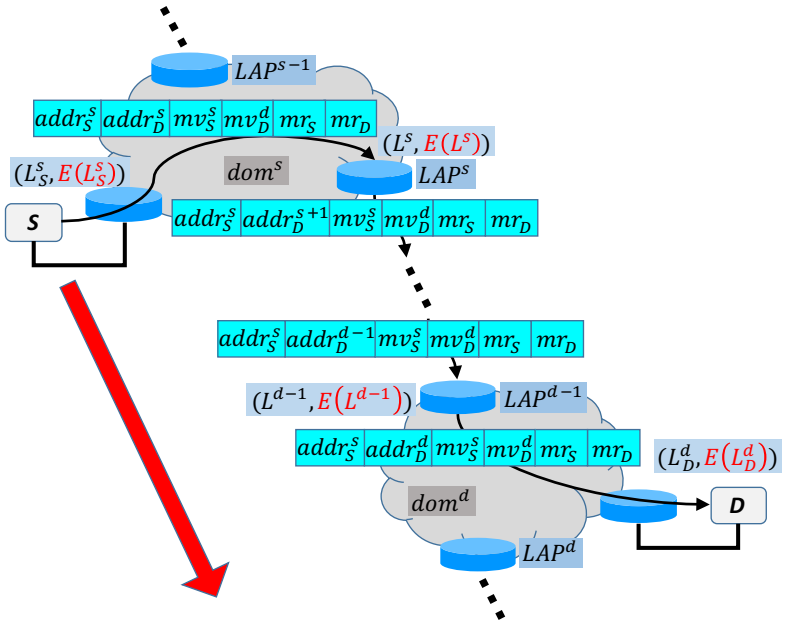


Figure 5.13: Top-down packet forwarding in BPF-HiLLIS.

3. Eventually, the packet arrives at LAP^{d-1}. According to the masking type mt_D^d, LAP^{d-1} queries MS^d for D's d-level address $addr_D^d$ containing the d-level unmasked or masked locator of the edge node to which the destination endpoint is connected. After replacing the destination address with $addr_D^d$, the packet is forwarded to the destination endpoint in the semi- or fully masked manner according to the masking type mt_D^d.

5.2.7.3 Bottom-up packet forwarding

If dom^d is a parent domain of dom^s with $d < s$, the source endpoint S obtains D's masking vector mv_D^d and d-level address $addr_D^d$. According to the masking type mt_D^d, the destination address contains the d-level unmasked or masked locator of the edge node to which the destination endpoint is connected. Thus, the source endpoint generates the packet

$$< mr_D \mid mr_S \mid mv_D^d \mid mv_S^s \mid addr_D^d \mid addr_S^s >.$$

For forwarding the packet to the destination endpoint, it is proceeded as follows (see Figure 5.14):

1. The packet is forwarded to LAP^{s-1} by means of the unmasked or masked default route according to mt_D^d.

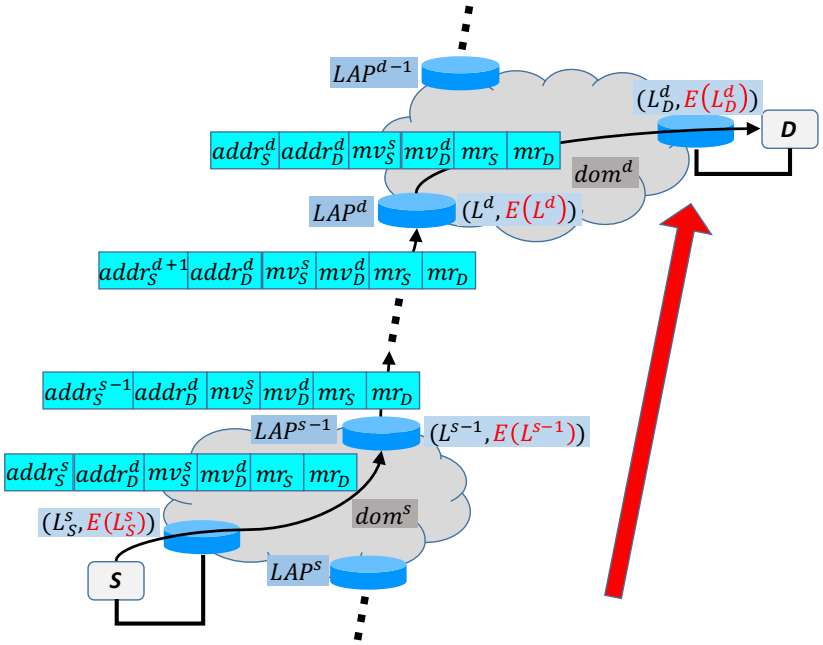


Figure 5.14: Bottom-up packet forwarding in BPF-HiLLIS.

2. For the incoming packet with $s-1 \leq k < d$, LAP^k checks whether the masking type mt_S^k is set.

   - ◇ If this is the case, LAP^k replaces the source locator of the packet with its k-level masked locator.
   - ◇ Otherwise, the source locator of the packet is replaced with the k-level unmasked locator of LAP^k.

   Thus, the source address of the packet is substituted with S's k-level address $addr_S^k$, and LAP^k gets the packet

   $$< mr_D \mid mr_S \mid mv_D^d \mid mv_S^s \mid addr_D^d \mid addr_S^k >.$$

   After that, the packet is forwarded to LAP^{k-1} by using the unmasked or masked default route according to the masking type mt_D^d.

3. Eventually, LAP^d receives the packet. According to the masking type mt_S^d, LAP^d replaces the source locator of the packet with its own d-level unmasked or masked locator. After that, the packet is forwarded to the destination endpoint in the semi- or fully masked manner on the basis of the masking type mt_D^d.

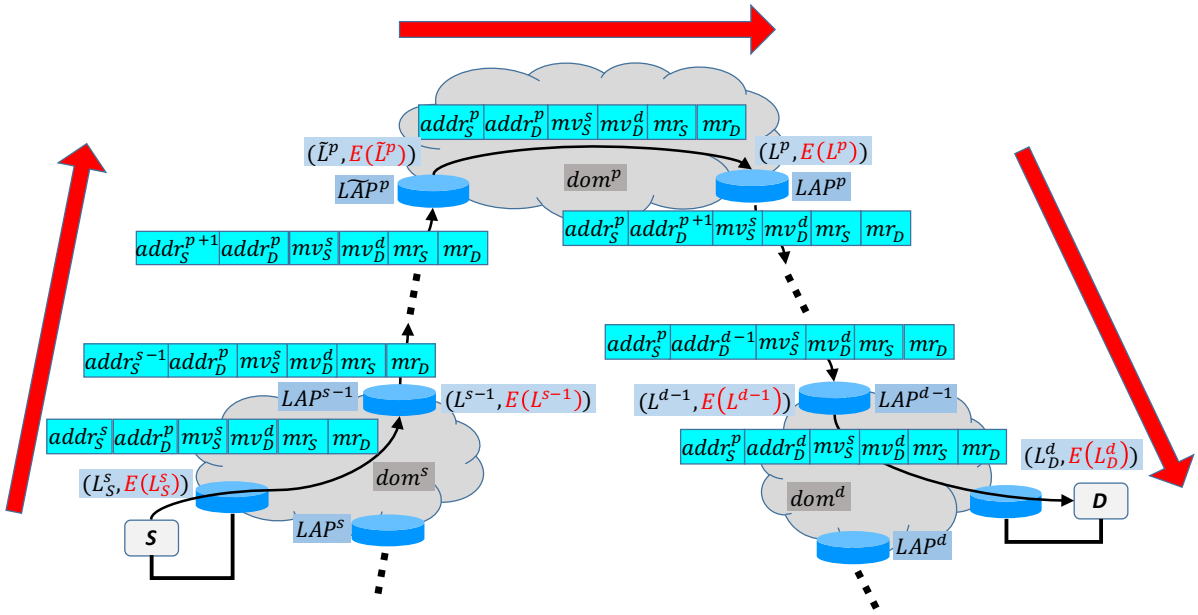


Figure 5.15: Packet forwarding between sibling domains in BPF-HiLLIS.

5.2.7.4 Packet forwarding between sibling domains

If dom^s and dom^d are sibling domains and dom^p is their next common parent domain with $1 \leq p < s$ and $1 \leq p < d$, the source endpoint S gets D's masking vector mv_D^d and p-level address $addr_D^p$. According to the masking type mt_D^p, this address contains the p-level unmasked or masked locator of LAP^p via which to reach dom^d. Thus, the source endpoint creates the packet

$$< mr_D \mid mr_S \mid mv_D^d \mid mv_S^s \mid addr_D^p \mid addr_S^s >.$$

The packet is forwarded in the following steps (see Figure 5.15):

1. Until the packet arrives in dom^p, the packet is forwarded in the bottom-up manner as described in Section 5.2.7.3. After replacing the source address of the packet with S's p-level address according to the masking type mt_S^p, $\widetilde{LAP}^p$ gets the packet

$$< mr_D \mid mr_S \mid mv_D^d \mid mv_S^s \mid addr_D^p \mid addr_S^p >.$$

2. On the basis of the masking type mt_D^p, the packet is forwarded to LAP^p in the semi- or fully masked manner within dom^p.

3. From then on, the packet is forwarded in the top-down manner as proposed in Section 5.2.7.2.

5.2.8 Fully blind packet forwarding on demand

By means of masking registration, an endpoint D can implicitly signal in which mode at which level a packet addressed to itself has to be forwarded. The endpoint D can register different masking ranks so that it can communicate with multiple endpoints according to various masking ranks. Here, a source endpoint S aiming to communicate with D selects a masking rank and checks by means of masking lookup whether the masking rank is registered. If not, S tries the lookup with another masking rank or with the lowest masking rank which is registered by default and indicates semi-masking in each domain on the route to D. In summary, the endpoint D determines which masking ranks can be applied to packets addressed to itself.

However, it is also desirable that S is capable of requesting full masking at selected levels. For that, we propose an approach described on the basis of sibling domain communication which includes intra-domain, top-down, and bottom-up packet forwarding (see Section 5.2.7). Thus, the endpoints S and D are located in dom^s and dom^d, and dom^p is their next common parent domain with $1 \leq p < s$ and $1 \leq p < d$.

We use approach 2 in case 3 (see Section 5.2.6.3) in order to set up a blind route between two selected nodes. For setting up blind routes within domains at selected levels between the endpoints S and D, it is proceeded as follows (see Figure 5.16):

1. S encrypts the destination identifier with D's public key assigned to the lowest masking rank and gets $E(ID_D)$. After that, S performs a lookup according to this masking rank as proposed in Section 5.2.4. Thus, S gets the p-level unmasked locator L^p of LAP^p via which dom^d can be reached.

2. After defining the masking rank and vector (mr_S, mv_S^s) for itself, S registers them at the mapping systems up to the level p as described in Section 5.2.3.

3. S defines the masking rank and vector (mr_D, mv_D^d) for D and sends the p-level *masking request* to its edge node:

$$masking\text{-}req^p\{(mr_D, mv_D^d, (L^p, E(ID_D)), (mr_S, mv_S^s, addr_S^p)\}.$$

   Here, $addr_S^p$ is S's p-level address which is registered at MS^p in step 2.

4. For the incoming masking request, the edge node checks the masking type mt_S^s in the masking vector mv_S^s:

   - ◇ If $mt_S^s = 1$, the edge node initiates the setup of a blind route between itself and LAP^{s-1} which is the LAP used for default routing to the next parent domain.

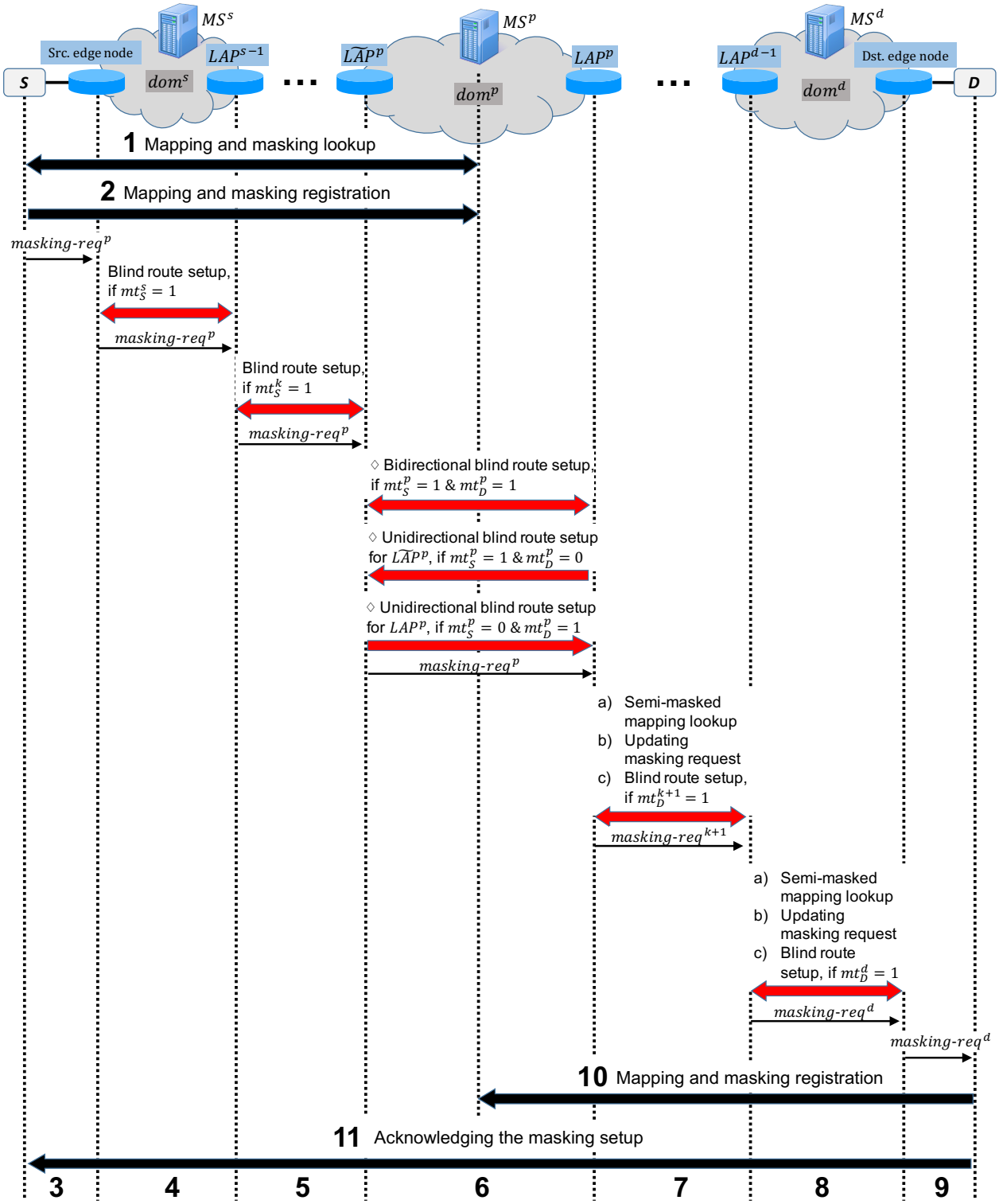


Figure 5.16: Masking setup on demand in BPF-HiLLIS.

After setting up the blind route, the masking request is sent to LAP^{s-1} by using the default route.

- ◇ Otherwise, the masking request is directly sent to LAP^{s-1} by means of the default route.

5. For the incoming masking request with $s-1 \leq k < p$, LAP^k checks whether the masking type mt_S^k is set:

   - ◇ If this is the case, a blind route is set up between LAP^k and LAP^{k-1}, and the masking request is sent to LAP^{k-1}.
   - ◇ Otherwise, LAP^k directly sends the masking request to LAP^{k-1} by using the default route.

6. Eventually, the masking request arrives at $\widetilde{LAP}^p$ which connects dom^{p+1} and dom^p with each other. This LAP checks the masking types mt_S^p and mt_D^p in the masking vectors mv_S^s and mv_D^d:

   - ◇ If both masking types are set, a bidirectional blind route is set up between $\widetilde{LAP}^p$ and LAP^p whose p-level unmasked locator is specified in the masking request. After that, the masking request is sent to LAP^p.
   - ◇ If $mt_S^p = 1$ and $mt_D^p = 0$, only a unidirectional blind route for $\widetilde{LAP}^p$ is set up, and the request is sent to LAP^p. Here, $\widetilde{LAP}^p$ puts an additional parameter into the blind route setup message, which signals that LAP^p does not have to send a blind route setup message for itself.
   - ◇ In case of $mt_S^p = 0$ and $mt_D^p = 1$, $\widetilde{LAP}^p$ sends the masking request to LAP^p. Here, the masking request contains the p-level unmasked locator of $\widetilde{LAP}^p$ and an additional parameter signalling that LAP^p has to induce to set up a unidirectional blind route for itself in the direction of $\widetilde{LAP}^p$.
   - ◇ Otherwise, the masking rank is directly sent to LAP^p.

7. For the incoming k-level masking request at LAP^k with $p \leq k < d-1$, it is proceeded as follows:

   (a) LAP^k resolves D's masked identifier to the $(k+1)$-level unmasked locator L^{k+1} of LAP^{k+1} via which to reach dom^d.

   (b) L^k in the masking request is replaced with L^{k+1}, and LAP^k gets the $(k+1)$-level masking request

$$masking\text{-}req^{k+1}\{(mr_D, mv_D^d, (L^{k+1}, E(ID_D)), (mr_S, mv_S^s, addr_S^p)\}.$$

(c) If mt_D^{k+1} is set, a blind route is set up between LAP^k and LAP^{k+1}, and the masking request is sent to LAP^{k+1}. Otherwise, LAP^k directly sends the masking request to LAP^{k+1}.

8. Upon receipt of the $(d-1)$-level masking request, LAP^{d-1} proceeds analogously. But here, LAP^{d-1} gets D's d-level unmasked edge locator L_D^d, and a blind route is set up between LAP^{d-1} and the edge node to which D is connected, if mt_D^d is set.

9. For the incoming masking request, D's edge node resolves $E(ID_D)$ to D's MAC address and sends the request to D.

10. After receiving the request, D registers the masking rank and vector (mr_D, mv_D^d) at the mapping systems up to the level p.

11. Finally, D generates a BPF-HiLLIS network packet addressed to S. As the source and destination masking ranks and vectors as well as the destination address of the packet, D takes the values from the masking request. The source address of the packet is D's d-level address registered in step 10. After that, the packet is sent to S as described in Section 5.2.7.4. This packet signals the acknowledgment of the masking setup in its payload.

After receiving the acknowledgment packet, the source endpoint S can start to send packets whose source and destination addresses are masked on the basis of the masking vectors mv_S^s and mv_D^d.

5.3 Analysis

In BPF-HiLLIS, the address of an endpoint consists of a locator and the identifier of the endpoint. As described in Section 5.2.1, it depends on the domain within which the endpoint has to be addressed, which locator has to be taken for the address. Moreover, the current level locator of the address is replaced with the next level locator at each level border. In this way, the length of the address is not dependent on the level of the domain in which the endpoint is located. Thus, the address length remains constant unlike in BPF-HAIR.

The identifier part of the address is encrypted with the associated public key of the endpoint according to the selected masking rank to be applied for the address. If the address has to be fully masked within a domain, the locator of the address is encrypted with the public key of the owner of the locator. Thus, it is well-defined, which parts the address consists of, and which part has to be encrypted with which public key. Hence, BPF-HiLLIS fulfils the requirement for a hierarchical addressing structure.

By means of mapping and masking lookup, a source endpoint directly gets the corresponding masked locator of a destination endpoint from the mapping system. For packet generation, the source endpoint thus needs only the corresponding public key of the destination endpoint according to the selected masking rank. The public key of an edge node is maintained by the DHCP server in the local network for which the edge node is responsible, where the edge node and the DHCP server are under the control of the local network provider. Moreover, the public key of LAP^k connecting dom^k and dom^{k+1} with each other is kept by MS^{k+1}. Here, LAP^k and MS^{k+1} are managed by the provider of dom^{k+1}. Hence, an additional infrastructure is not required for exchanging and certifying the public keys of network nodes and LAPs.

BPF-HiLLIS adopts the hierarchical scheme of BPF-HAIR, in which routing table entries are already aggregated accordingly. In this way, the entire infrastructure is super- and subnetted by design. Thus, an additional process is not required to aggregate and partition the networks. Moreover, BPF-HiLLIS extends the hierarchical scheme in a way that endpoints can be attached to nodes at an arbitrary level. In summary, we can state that BPF-HiLLIS fulfils all of the architectural requirements for an adequate BPF design.

5.3.1 Semi-blindness

If the masking vector for a packet address indicates a semi-masking within a domain at a specified level, the identifier part of the address is transferred and processed end-to-end in encrypted form, while the address locator is handled within the domain in cleartext. Thus, semi-masking of a packet address within a domain provides network identifier confidentiality (NIC) for the address within the domain. In order to forward a packet whose destination address is semi-masked within a domain, the network nodes in the domain do not need to maintain masked routing table entries. Hence, a network node in the domain knows the previous and next hop of the packet.

As already discussed in Section 4.1.4.1, NIC deduces sender/recipient and relationship unlinkabilities for communicating endpoints, since only they are the authorised entities having access to the cleartext identifiers of packets transferred between them. Because of handling locators in cleartext, the unlinkability properties do not apply to domains and local networks.

In case of intra-domain communication, a packet can thus be linked to its source or destination local network, if the source or destination address of the packet is semi-masked. In case of inter-domain communication, a local adversary being located within an intermediate

domain can link a packet to its previous or next domain, if the source or destination address of the packet is semi-masked within the intermediate domain. To a strong and weak global adversary, the same unlinkability properties apply as in the semi-blind modes of BPF-GLI and BPF-HAIR.

5.3.1.1 Mapping system

A semi-masked mapping table entry for an endpoint maintains its identifier in encrypted form and the associated locator in cleartext. If an endpoint registers its semi-masked mapping at the mapping system in the domain in which the endpoint is located, the location of the local network, to which the endpoint is connected, is disclosed to the mapping system. But the mapping system does not know which endpoint is connected to that local network. In case of registering a semi-masked mapping of the endpoint at a mapping system at a higher level, the mapping system knows the attachment point of the next child domain via which to reach the endpoint. However, the mapping system does not know which endpoint can be reached via that child domain. Thus, NIC also applies to semi-masked mapping table entries.

If the address of the requesting endpoint is semi-masked in a mapping/masking lookup request transferred within a domain, the request cannot be linked to an endpoint as the requesting endpoint. But due to the cleartext locator of the address, it is disclosed within the domain from which child domain or local network the request originates.

5.3.2 Full blindness

In case of full masking a packet address within a domain, the identifier and locator parts of the address are transferred and processed within the domain in encrypted form. This provides NIC as well as network locator confidentiality (NLC) for the address within the domain. As already stated in Section 4.1.4.2, together NIC and NLC provide the sender/recipient and relationship unlinkabilities for communicating endpoints as wells as domains and local networks.

If the source or destination address of a packet is fully masked within a domain, a local adversary being located within that domain thus does not know, from or to which endpoint as well as domain and local network the packet originates or is addressed. Moreover, the same unlinkability properties apply against a weak and strong global adversary as are the case in the fully blind modes of BPF-GLI and BPF-HAIR.

As in BPF-GLI and BPF-HAIR, semi- and full masking of a packet address in BPF-HiLLIS do not aim to provide integrity of the address. In this regard, the same discussions also apply here as in Section 4.1.4.5.

5.3.2.1 Masked routing

In Section 5.2.5, we have discussed that a network node can make void the effect of the full blindness, if both the semi- and fully blind mode are applied in a domain. In order to resolve this issue, we have enhanced the masked routing as described in Section 5.2.5. Thus, network nodes in a domain maintain random distances to each other in their masked routing tables. In this way, the port-distance tuple in a masked routing table entry for the network node N does not match the port-distance tuple in a unmasked routing table entry for the same network node. Hence, a network node cannot disclose that a packet containing N's masked locator as its destination locator is addressed to N by exploiting its unmasked routing table.In this regard, both the semi- and full blindness can be applied within a domain.

For forwarding a packet with a fully masked destination address within a domain, network nodes in that domain have to maintain masked routing table entries. In principle, it is sufficient that each network node on the route of the packet within the domain keeps a masked routing table entry for the node to be taken by the packet as its next hop. In Section 5.2.6, we have proposed approaches for different cases to selectively set up masked routing table entries.

Section 5.2.8 has proposed an approach by means of which full blindness can be set up within domains at selected levels between two endpoints. During the masking setup phase, network nodes on the route between the endpoints only know to which local network and next domain the masking request is addressed. After the setup, the packet addresses are masked on the basis of the selected masking ranks for the source and destination endpoints. This means that the packet addresses are encrypted with the public keys assigned to the selected masking ranks. Since thus the byte values of the addresses in the masking request and network packets transferred between the endpoints differ from each other, the masking request and network packets cannot be correlated with each other.

5.3.2.2 Mapping system

If a mapping table entry for an endpoint is fully masked at a mapping system in a domain, the entry holds the identifier of the endpoint and the associated locator in encrypted form. In this way, information about endpoints as well as local networks and next child domains is masked from the mapping system. Thus, NIC and NLC apply to fully masked mapping table entries.

In case of full masking the address of the requesting endpoint in a mapping/masking lookup request transferred within a domain, the request cannot be linked to an endpoint as the requesting endpoint. Moreover, it is also masked within the domain from which child domain or

local network the request originates. If the requesting endpoint gets a fully masked address, the endpoint does not know to which local network the requested endpoint is connected, or via which domain the requested endpoint can be reached.

5.3.3 Blindness taxonomies

A masking vector for the address of an endpoint being located in a domain at level x is a bit vector of length x in big-endian order. The masking vector determines at levels 1 to x whether the semi- or full masking has to be applied for the address. Thus, up to 2^x different masking vectors can be defined for the address of the endpoint. The blindness level provided by a masking vector is specified by its masking rank. Below, we define two blindness taxonomies into which certain masking ranks are classified.

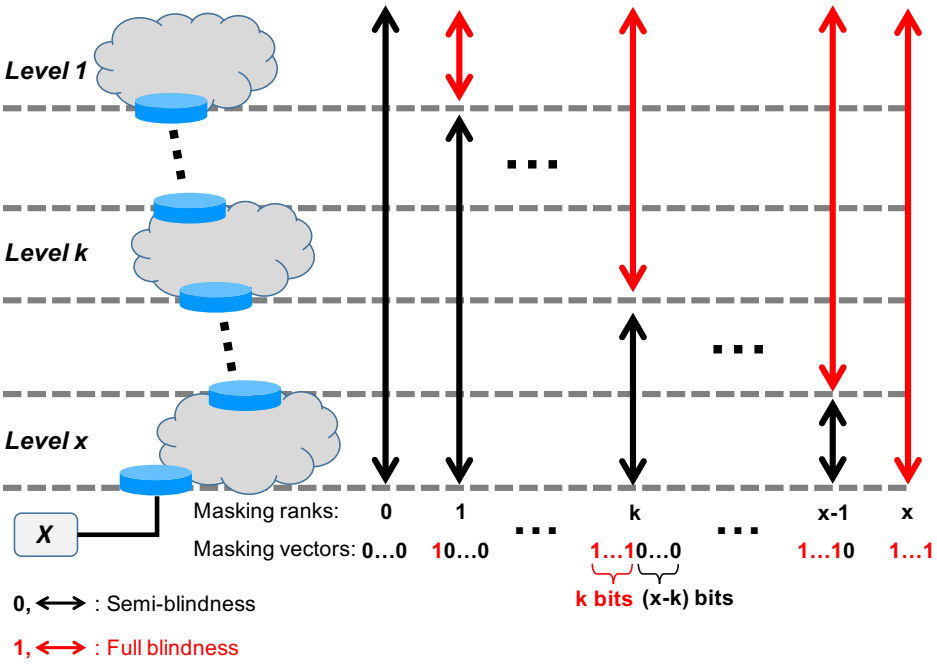


Figure 5.17: Blindness taxonomy 1 in BPF-HiLLIS.

5.3.3.1 Taxonomy 1

In this taxonomy, a masking rank determines how many bits are set beginning with the most-significant bit in the masking vector (see Figure 5.17). For the endpoint X in dom^x, the masking rank $mr_X = k$ with $0 \leq k \leq x$ specifies the masking vector

$$mv_X^x = (\underbrace{1...1}_{k\ bits} \overbrace{0...0}^{(x-k)\ bits}).$$

In case of performing this masking rank, the full blindness is effective for X's address within domains at levels 1 to k. Thus, NIC as well as NLC and their deduced unlinkability properties are the case within these domains. Moreover, the semi-blindness holds for the address within domains at levels being higher than k. Hence, only NIC and the sender/recipient and relationship unlinkabilities for the endpoint apply from level $(k+1)$. This taxonomy defines up to $(x+1)$ different masking ranks for the endpoint X. Here, the higher the masking rank being applied to X's address, the bigger is the radius of domains within which the address is fully masked, beginning with the Core.

Since only the semi-blindness is the case up to level $(k+1)$, adversaries within domains at these levels know from which previous domain packets sent by X originate, and to which next domain packets sent to X are addressed. Thus, such packets may be affected by active attacks performed against an entire domain, e.g., manipulating all packets originating from a certain domain. In this regard, the endpoint can select the semi-blindness within a domain, if the domain is protected against such attacks. Hence, the radius of domains regarded as secure becomes smaller by applying higher masking ranks in this taxonomy. On that score, this taxonomy is comparable with the security taxonomy in IPsec-VPN [YS01], where IP Encapsulating Security Payload (ESP) tunnel length becomes longer.

In this taxonomy, up to $((s+1) \times (d+1))$ different masking rank combinations are possible for the source and destination endpoints S and D being located in sibling domains dom^s and dom^d. Figure 5.18 illustrates the masking combinations for $s = 3$ and $d = 3$. Here, a unbroken black or red arrow denotes that the source and destination addresses of the packet are semi- or fully masked. Moreover, a dashed black-red arrow signifies that the source or destination address is semi- or fully masked. If the start point of the arrow is black and the end point is red, the source address is semi-masked and the destination address is fully masked. In case of colouring the start and end points inversely, the full and semi-masking is applied to the source and destination address.

By performing higher masking ranks, the communicating endpoints S and D initiates that the respective previous and next domain of a packet is masked within more intermediate domains. In case of symmetrically increasing the masking ranks for the source and destination addresses, we can observe that the scope of the full blindness grows in direction of both endpoints, beginning with their next common parent domain (see the masking rank combinations $(0,0), (1,1), (2,2), (3,3)$ in Figure 5.18). If we perform, for example, the masking rank 2

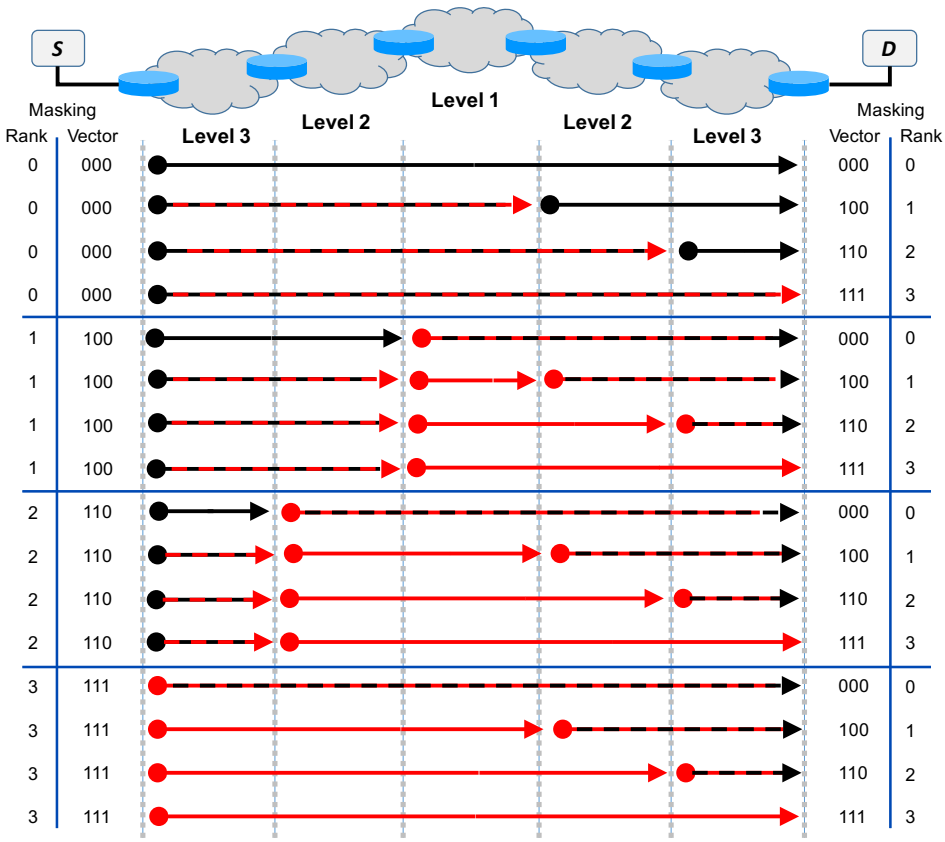


Figure 5.18: Masking rank combinations in blindness taxonomy 1.

for the source address and increase the masking rank for the destination address, the full blindness for both endpoints begins at level 2 on the source side and grows in direction of the destination endpoint. Moreover, the blindness scopes are reversed in two mirror-inverted masking rank combinations such as in $(0, 2)$ and $(2, 0)$.

5.3.3.2 Taxonomy 2

In contrast to the previous taxonomy, a masking rank in this taxonomy determines how many bits are set beginning with the most-least bit in the masking vector (see Figure 5.19). For the endpoint X in dom^x, the masking rank $mr_X = k$ with $0 \leq k \leq x$ specifies the masking vector

$$mv_X^x = (\overbrace{0...0}^{(x-k)\ bits}\ \underbrace{1...1}_{k\ bits}).$$

If we use this masking rank, the full blindness for X's address begins from the level $(x - k)$. Thus, NIC as well as NLC and their deduced unlinkability properties apply to the endpoint X

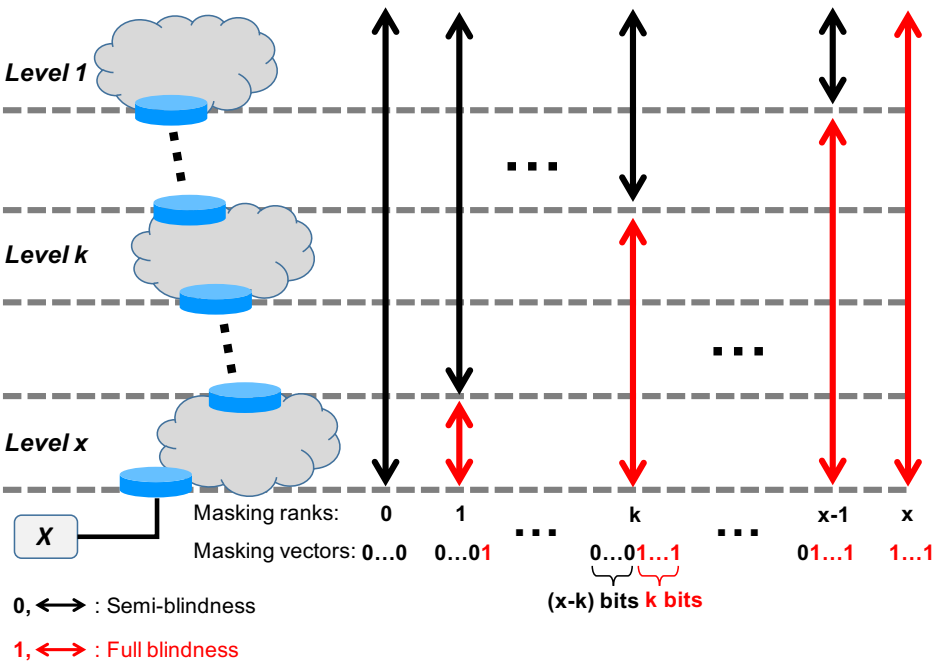


Figure 5.19: Blindness taxonomy 2 in BPF-HiLLIS.

within domains at levels being higher than $(x-k)$. Moreover, the semi-blindness are the case at levels 1 to $(k-1)$. Hence, only NIC and its unlinkability properties hold for the endpoint X within domains at these levels. As the previous taxonomy, this taxonomy defines up to $(x+1)$ different masking ranks for the endpoint X.

By applying higher masking ranks, the radius of domains becoming fully blind grows beginning with the domain in which the endpoint is located. By means of the masking rank 1, the endpoint masks the location of its edge node from the domain provider. In case of the masking rank 2, the endpoint additionally masks the location of its domain from the provider of the parent domain. In this way, the endpoint can mask its respective location within higher parent domains in direction of the Core. Thus, the endpoint increases the radius of domains within which the full blindness applies to the endpoint.

As the previous taxonomy, this taxonomy also defines up to $((s+1) \times (d+1))$ different masking combinations for the source and destination endpoints S and D which reside in sibling domains dom^s and dom^d. Figure 5.20 illustrates the masking combinations for $s = 3$ and $d = 3$. Here, we have used the same denotation as in Section 5.3.3.1.

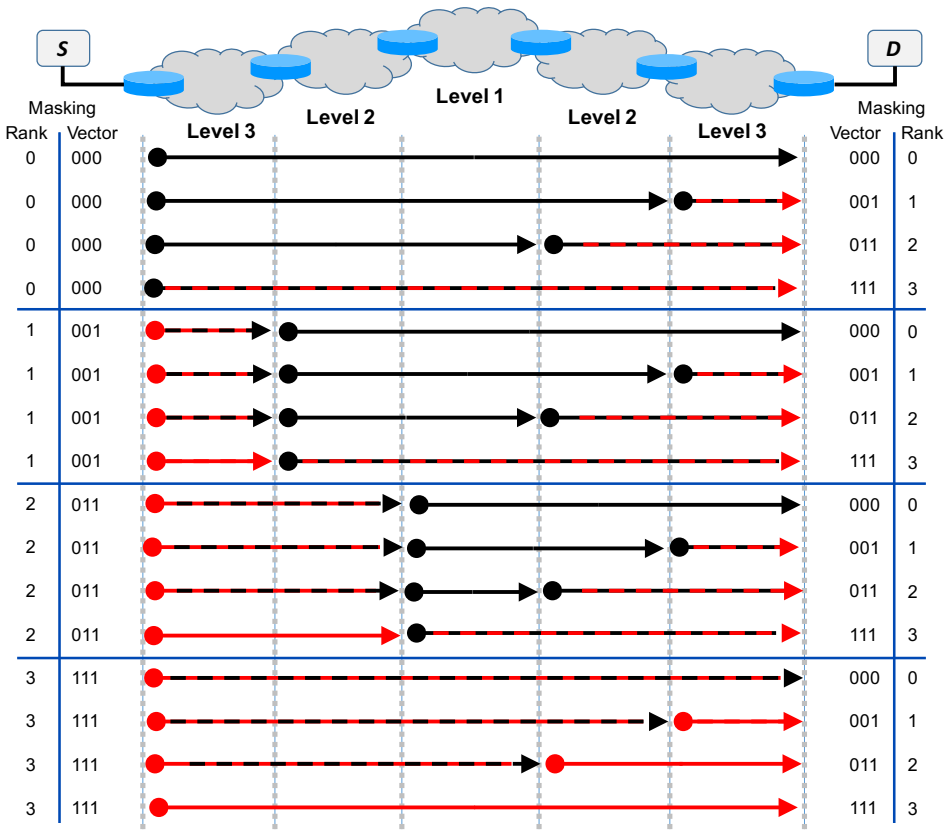


Figure 5.20: Masking rank combinations in blindness taxonomy 2.

If the communicating endpoints S and D symmetrically increase the masking ranks, the scope of the semi-blindness for both endpoints decreases from the source and destination sides in direction of their next common parent domain (see the masking rank combinations $(0,0),(1,1),(2,2),(3,3)$ in Figure 5.20). In the combinations $(1,1)$ and $(2,2)$, the full blindness does not apply to both endpoints at once. Thus, the full blindness holds only for the source endpoint in the ultimate source domain and its parent domain, while only the destination endpoint gets the full blindness in the ultimate destination domain and its parent domain. In order to achieve the simultaneous full blindness for both endpoints within one or multiple domains, one of the communicating endpoints has to apply the masking rank 3, while the other one has to use at least the masking rank 1. As in the previous taxonomy, the blindness scopes are reversed in two mirror-inverted masking rank combinations.

Cross combinations of the masking ranks from the blindness taxonomies 1 and 2 are also possible and illustrated in Figure 5.21. If we perform the masking rank 1 from taxonomy 1 for the source address and the masking rank 2 from taxonomy 2 for the destination address, we have the simultaneous semi- and full blindness within the ultimate source and destination domains as well as their parent domains. Moreover, an asymmetric blindness is the case

within the common parent domain. By means of the cross combinations, the simultaneous semi- and full blindnesses thus arise at once and within the ultimate source and destination domains as well as their parent domains. Additionally, we have then an asymmetric blindness within the common parent domain and its child domains.

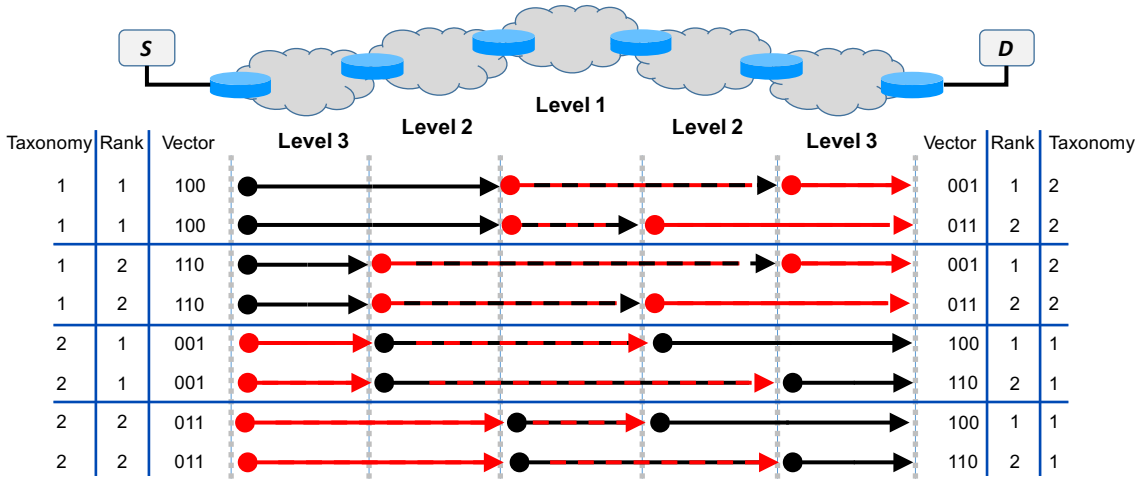


Figure 5.21: Cross combinations of the masking ranks from the taxonomies 1 and 2.

5.4 OLV-Openflow

In OpenFlow, a flow match field is based on the TLV format, and the types defined in OpenFlow rely on the IP packet structure. Thus, we cannot define flows in order to realise a clean-slate approach proposing another packet structure. One of the possible ways to implement a clean-slate approach by means of OpenFlow is handling packets at the controller hop-by-hop. For the prototype implementation of the basic BPF design, we have proceeded in this manner, which has introduced a considerable amount of overhead.

For resolving this issue, we have reinterpreted one of the already existing flow match field types in the implementations of BPF-GLI and BPF-HAIR. In this way, we could define flows which make it possible to utilise the flow-based packet forwarding in OpenFlow. However, packets still have to be sent to the controller at certain nodes, specifically at edge and gateway nodes. This is because the flows which we could define are not able to realise the required operations at these nodes. Thus, we could not leverage the benefits of OpenFlow in its entirety.

In principle, it is generally possible to extend OpenFlow by flow match field types required by clean-slate architectures. But the extensions would be highly architecture specific, and each

extension would introduce huge implementation costs. In this regard, we replace the TLV-based mechanism in OpenFlow with an OLV-based proceeding, which we call OLV-OpenFlow. This OpenFlow version defines a match field on the basis of an offset and length. Thus, OLV-OpenFlow is type-free. In this way, an approach defining another packet structure can be readily realised to leverage the SDN-like implementation and flow-based packet forwarding.

5.4.1 Construction

In OpenFlow 1.3, a packet arriving at a datapath is firstly parsed on the basis of pre-defined types defining the respective fields of the packet according to supported protocols. Next, the packet (i.e., its field values) goes through a pipeline in which flows matching the packet are determined as well as the instructions defined in these flows are performed on the packet. OLV-OpenFlow adopts this packet processing pipeline except the packet parsing step. Thus, a datapath does not have to be protocol aware anymore.

OLV-OpenFlow also adopts the OpenFlow 1.3 protocol via which a datapath communicates with a controller and the controller manages the datapath. Moreover, all of the structures defined in OpenFlow 1.3 remain the same in OLV-OpenFlow with the exception of the structures for flow match and instruction fields, which we describe below.

5.4.1.1 Flow match field

In OLV-OpenFlow, a flow match field consists of the following attributes:

[Offset | Length | Comparison operator | Object | Value].

Here, the Offset and Length are specified in bits. OLV-OpenFlow supports the comparison operators: *equal to*, *not equal to*, *greater than*, and *less than*. The Object determines whether the Value has to be compared to the packet, the metadata, or the ingress port number of the packet. The metadata is a register used by OpenFlow to carry information from a flow table to the next one.

If the Object in the match field of a flow is the incoming packet itself, a datapath firstly takes the field from the packet which is specified by the Offset and Length in the flow match field. Next, the datapath compares the value of this field with the Value in the match field. If both values match, the instruction defined in the flow is performed. In case the Object is the metadata of the packet, the datapath takes the correspondent field from the metadata and proceeds in the same manner. If the Object specifies the ingress port number of the packet, the Value is compared to the number of the port via which the packet has arrived.

5.4.1.2 Flow instruction

In OpenFlow, a flow contains one or multiple instructions. An instruction can modify pipeline processing such as directing the packet to another flow table or contains actions which are performed either immediately or after exiting the processing pipeline. The type-free actions (e.g., sending the packet via a specified port) remain the same in OLV-OpenFlow. The type-related actions (e.g., pushing a VLAN tag) are replaced with the following ones:

Manipulating the value of a field
This action consists of the following attributes:

[Offset | Length | Operator | Value].

Here, the Offset and Length are specified in bytes and determine the field from the matching packet which has to be manipulated with the Value on the basis of the Operator. In this way, the value of the field from the packet can be overwritten with the Value. Moreover, OLV-OpenFlow supports the operators: *addition*, *subtraction*, *and*, *or*, *xor*, *not*, *left-shift*, and *right-shift*.

Adding and deleting a field
By means of this action, a field from the packet can be removed, or a new field can be added to the packet. Here, the field is specified by a offset and length.

Replacing two fields with each other
This action replaces the values of two fields from the packet with each other. Here, both fields have the same length and are specified with two offsets and the length.

5.4.2 Implementation

For the implementation of OLV-OpenFlow on the switch and controller sides, we have utilised the OpenFlow 1.3 user-space software switch implementation [CPq16b] and the OpenFlow controller NOX implementation [CPq16a] as the basis code. Here, we have reimplemented both realisations by replacing the type-related parts with the type-free structures described above. The OLV-OpenFlow compatible controller is called *OLV-NOX*.

5.5 Implementation

In the realisation of BPF-HiLLIS, two data structures implement semi- and fully masked addresses. In each of them, an attribute defines the level of the locator. For BPF-HiLLIS,

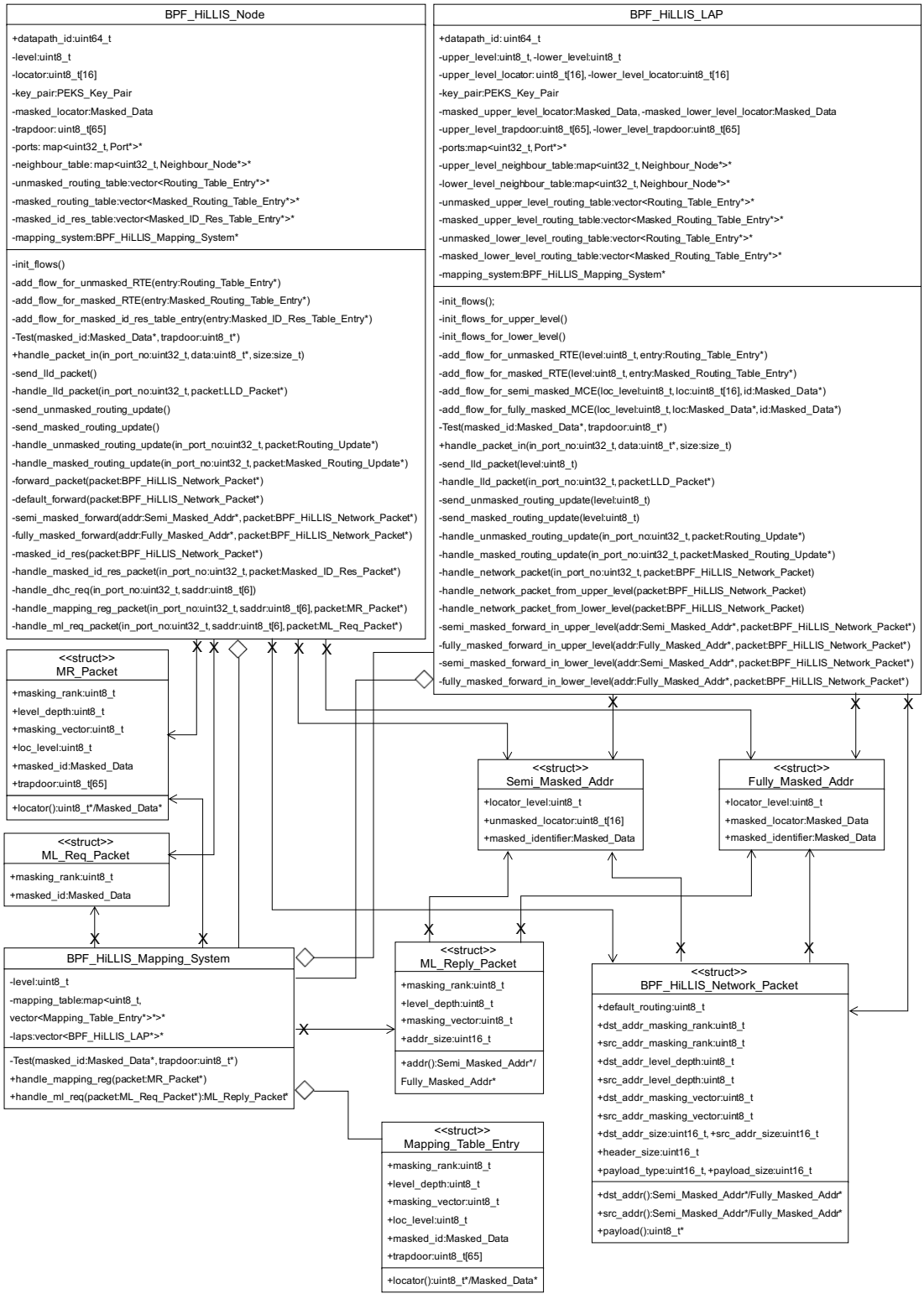


Figure 5.22: UML diagram of *BPF_HiLLIS_Node* and *BPF_HiLLIS_LAP*.

we have implemented a network packet header structure according to equation 5.3 (see Figure 5.22). Here, we have omitted the last address field as in the implementations of BPF-GLI and BPF-HAIR. Moreover, a flag in our header structure states whether the packet has to be forwarded by using the default route. Furthermore, two attributes define the levels of the domains within which source and destination endpoints are located. Each of the source and destination masking vectors has one byte. By means of this implementation, we can thus realise a topology consisting of up to eight levels.

Since we can realise up to eight levels, a masking rank belonging to taxonomy 1 can have one of the natural numbers 10,...,18. Here, the first digit represents the taxonomy, and the second one determines the number of bits to be set in the associated masking vector according to the taxonomy. Analogously, taxonomy 2 provides the masking ranks 20,...,28.

Multiple new packet structures have been implemented to realise the mapping and DHCP functionalities in BPF-HiLLIS. For the link layer discovery, masked identifier resolution and unmasked/masked routing, the same packet structures have been used as in the implementations of BPF-GLI and BPF-HAIR. Each of these packet types gets a new Ethernet payload type in our implementation. For encryption of locators and identifiers as well as for generation of key pairs and trapdoors, we have leveraged the PEKS library [ALPK07].

The BPF-HiLLIS functionalities on the host side have been realised by the C++ class *BPF-HiLLIS-BNS* implemented in the same manner as in the implementations of BPF-GLI and BPF-HAIR. The framework *BPF-HiLLIS-BNS* interacts with applications via the interface *BPF_HiLLIS_BNS_Socket* in the same way as in the realisations of BPF-GLI and BPF-HAIR.

For realising the BPF-HiLLIS functionalities on the network side, we have expanded the OLV-OpenFlow controller OLV-NOX by the component *BPF_HiLLIS*. Here, the C++ classes *BPF_HiLLIS_Node* and *BPF_HiLLIS_LAP* implement the functionalities of a network node and LAP. An UML diagram of both classes and the associated data structs is given in Figure 5.22. Here, we have omitted the data structures whose UML diagram is already presented in Figures 3.9, 4.12, and 4.35.

As in the implementation of BPF-HAIR, a new controller component is started for each domain. Here, network nodes and LAPs of each domain are managed by the controller component started for that domain. For each node and LAP in a domain, a new object is created from the classes *BPF_HiLLIS_Node* and *BPF_HiLLIS_LAP*. Moreover, the controller component for that domain identifies and manages the objects representing nodes and LAPs in the

same manner as proposed for the implementations of BPF-GLI and BPF-HAIR.

5.5.1 Packet processing pipeline

For each node and LAP in a domain, the associated controller component defines flows sending link layer discovery, routing update as well as mapping registration and lookup packets to the controller. Additionally, each network node maintains further flows sending DHCP and identifier resolution packets to the controller. These flows are maintained in the first flow table (default flow table) at each node and LAP. The offset and length in the match field of such a flow is the offset and length of the EtherType. Moreover, the match field contains one of the values defined for the packet types above as the value of the match field. The flow has a single instruction consisting of the action *Controller-Output*. In this way, packets of such types are sent to the controller for handling them in the SDN-like manner, which we discuss below.

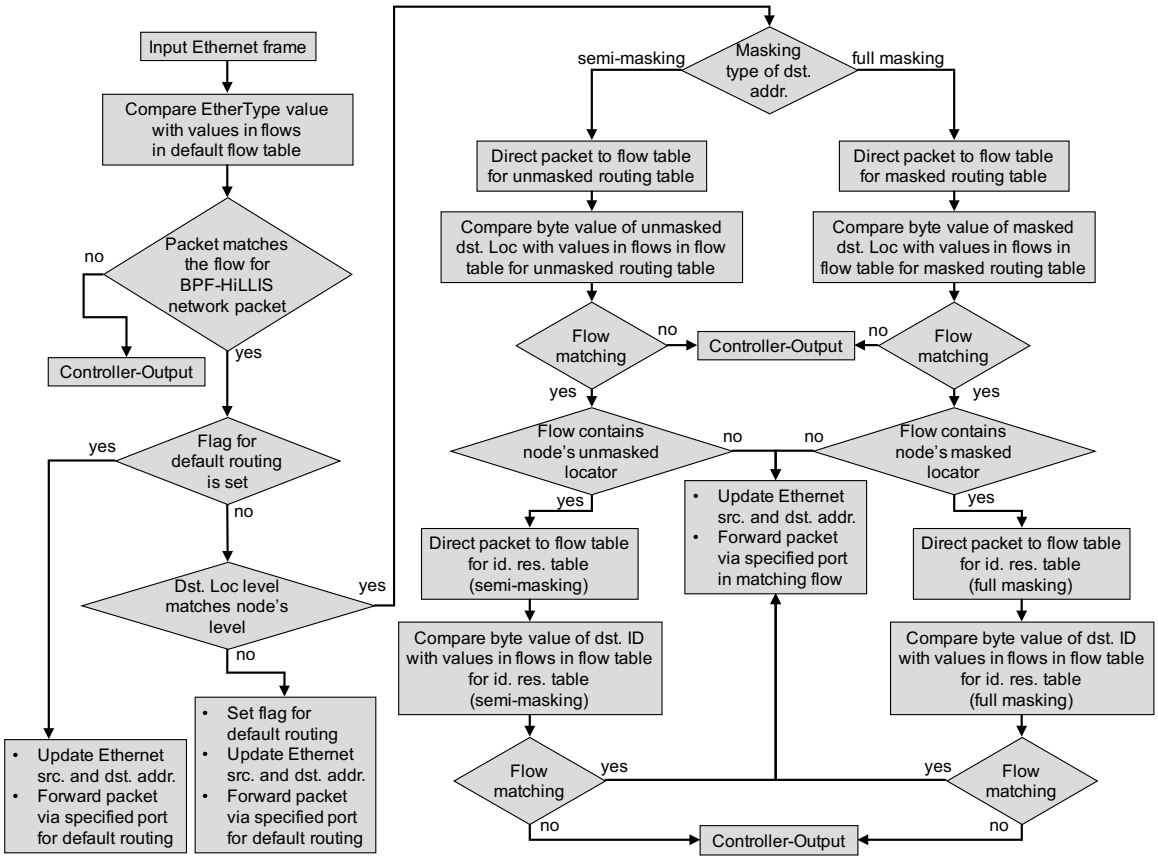


Figure 5.23: Flowchart for packet processing pipeline at network node.

5.5.1.1 Network node

The packet processing pipeline at a network node consists of the multiple flow tables. The flowchart of this processing pipeline is given in Figure 5.23. Besides the flows described above, the network node maintains a further flow in its default flow table. This flow checks whether the flag for default routing is set, and forwards the matching packets via the specified port. Moreover, a further flow in the default flow table checks the level of the current destination locator in an incoming packet. If the level does not match the level of the node, the flow sets the flag for default routing and forwards the packet via the port specified for default routing.

The controller component defines a further flow for the node in its default flow table to check the current masking type for the destination address of an incoming packet which does not match the two flows defined above. If the destination address is semi-masked, the packet is directed to the flow table containing the flows for unmasked routing table entries. In case of full masking, the flow tells the packet to go on with the flow table for masked routing table.

In each of the flow tables for unmasked and masked routing tables, one flow is defined with the routing information for the node itself. If the packet matches one of these two flows, i.e. the packet is addressed to the node, the packet is directed to a further flow table holding the flows for identifier resolution table entries. The packet is sent to the controller, if no flow in the respective flow table matches the packet. Thus, an incoming network packet goes through a pipeline consisting of two further flow tables which are discussed in detail below.

Flow tables for routing tables

The unmasked and masked routing table setups are implemented in the same SDN-like manner as in BPF-GLI and BPF-HAIR. For a newly created or updated routing table entry for the node N_i at the controller component, the network node N creates a new flow or to update the flow for the entry in the flow tables for the unmasked and masked routing tables.

If the entry is masked, the offset and length of the flow match field is the offset and length of the locator part in the fully masked destination address of a BPF-HiLLIS network packet, and the value is the entire byte value of $E(Loc_{N_i})$. Moreover, the flow has a single instruction having the action *Output* with the port number specified in the routing table entry for N_i. Additionally, the instruction contains actions updating the Ethernet source and destination addresses of matching packets. In this way, packets having N_i's masked locator as their destination locator match the flow and are forwarded via the specified port.

The offset and length of the unmasked flow for the node N_i is the offset and length of the unmasked locator in the semi-masked destination address, and the value is the byte value of Loc_{N_i}. As in the masked flow, the actions in the unmasked flow induces to update the

Ethernet source and destination addresses as well as to forward the matching packets via the port whose number is specified in the unmasked routing table entry for N_i.

Flow tables for identifier resolution table

For the identifier resolution cache table entry for the host X, the controller component tells the network node N to create a new flow or to update the existing one for the entry. Such a flow is put temporarily. Since unmasked and masked locators have different sizes, the destination identifiers have different offsets in network packets addressed to host X in the semi- and fully masked manner.

Because of this reason, two flows with different offsets in their match fields have to be defined in two different flow tables for the host X. A packet is directed from the flow tables for the unmasked and masked routing tables to these flow tables. The offsets in the match fields of these flows are the offsets of the identifier parts in the semi- and fully masked destination addresses. The match fields of both flows have the same byte length and value of $E(ID_X)$. Moreover, both flows contain actions to update the Ethernet source and destination addresses and to forward matching packets to the specified port.

In this way, only the first packet for a communication endpoint pair has to be sent to the controller by the destination edge node. After this packet, a corresponding flow for the destination endpoint is defined. Thus, the remaining packets match this flow and do not have to be sent to the controller anymore.

5.5.1.2 Level Attachment Point

The packet processing pipeline at a LAP also consists of multiple flow tables, and its flowchart is given in Figure 5.24. Here, a flow in the default flow table checks whether an incoming packet has arrived from a domain at the lower or upper level, and directs the packet to the flow table for lower or upper level. In two further flow tables, a LAP maintains flows for semi- and fully masked mapping cache entries. These flow tables are discussed below.

Four flow tables are maintained for the unmasked and masked routing tables at the upper and lower levels. Flows in each of these flow tables are defined in the same manner as the flows for the routing table entries at a network node. In each of the flow tables for the routing tables at the upper level, a further flow checks the destination locator level. If the level is not equal to the upper level, the flag for default routing is set, and the packet is forwarded via the port specified for default routing. In each of these flow tables, a further flow sends the packet to the controller, if no flow in the respective flow table matches the packet.

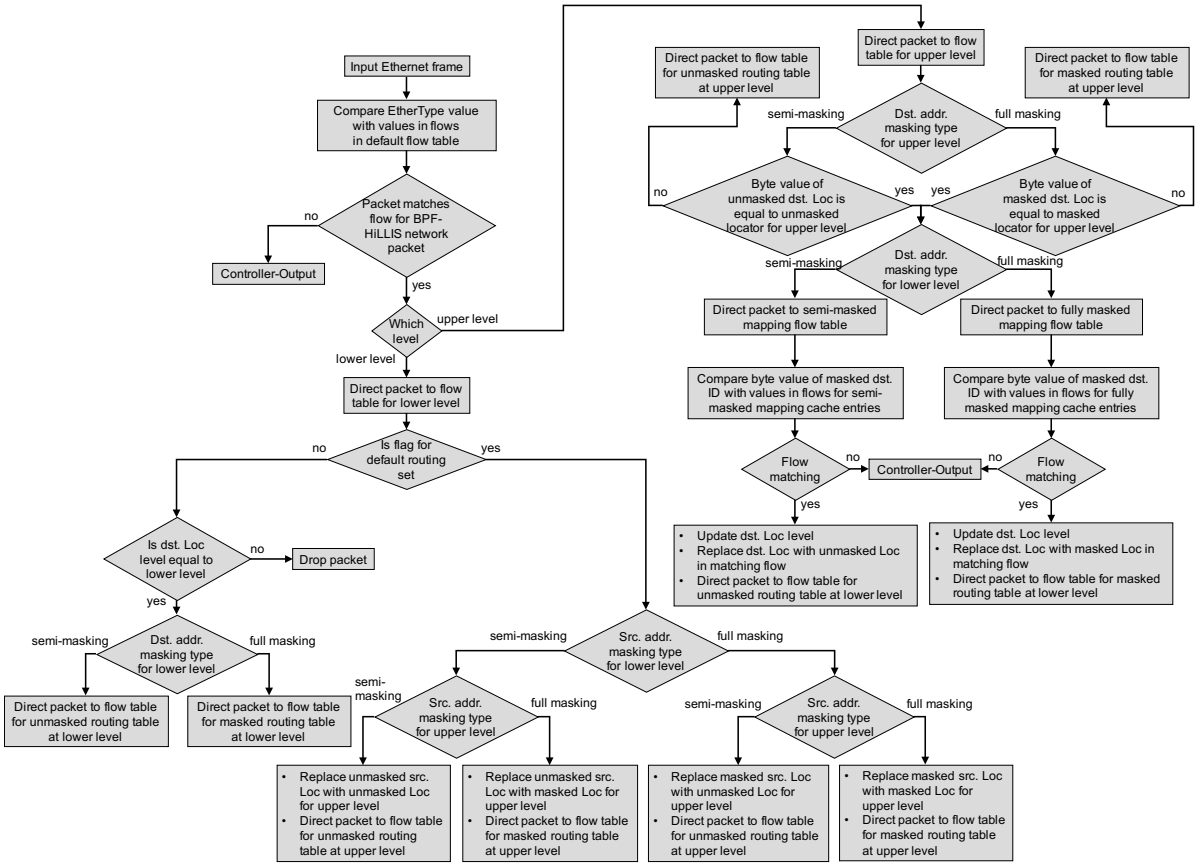


Figure 5.24: Flowchart for packet processing pipeline at LAP.

Flow table for the lower level

A packet arriving from the domain at the lower level is directed to this flow table. A flow in this flow table checks whether the flag for default routing is set. Moreover, the flow checks the destination locator level and the destination address masking type of the incoming packet. If the flag is not set as well as the level is the lower level and the destination address is semi-masked, the packet is directed to the flow table for the unmasked routing table at the lower level. In the same manner, a further flow directs the packet to the flow table for the masked routing table at the lower level, if the masking type is the full masking. Thus, the packet is forwarded within the domain at the lower level.

If the flag for default routing is set, the packet is directed to one of the flow tables for upper level routing tables. Before that, the source locator has to be replaced with the upper level locator of the LAP. For that, the corresponding flows have to look first into the destination address masking type, since the offset of the source locator changes according to this masking type. After that, the source address masking types for the lower and upper levels are checked. In this way, the matching flow is determined which substitutes the unmasked or masked source

locator with LAP's unmasked or masked locator for the upper level and directs the packet to the flow tables for unmasked or masked routing tables at the upper level. Thus, the packet is forwarded within the domain at the upper level.

Flow table for the upper level

If the packet has come from the domain at the upper level, the packet is directed to this flow table. After checking the masking type of the destination address, a flow in this flow table determines whether the destination locator has the same byte value as LAP's unmasked or masked locator for the upper level. If this is not the case, the packet is directed to one of the flow tables for upper level routing tables in order to forward the packet back within the domain at the upper level.

If the byte values of the destination locator and LAP's unmasked or masked locator for the upper level are the same, the destination address masking type for the lower level is checked. After that, the packet is directed to one of the flow tables for semi- and fully masked mapping cache entries.

Flow tables for mapping cache entries

After a semi- or fully masked mapping lookup, a LAP caches the mapping. For a semi- or fully masked mapping cache entry containing the masked identifier of the endpoint X, a temporary flow is defined in the semi- or fully masked mapping flow table. The offset and length of the flow match field is the offset and length of the identifier part in the semi- or fully masked destination address of a BPF-HiLLIS network packet. Moreover, the flow match field value is the byte value of $E(ID_X)$. The flow contains three actions. The first two actions updates the destination locator level and the destination locator with the level and locator from the mapping cache entry. The last one directs the packet to one of the flow tables for lower level routing tables.

In this regard, the first packet for a communicating endpoint pair is sent to the controller by LAPs on the destination side in order to perform the mapping lookup. Since an associated flow is created after the first packet, the remaining packets match the flow and do not have to be sent to the controller anymore.

5.5.2 DHCP and Mapping

The mapping functionalities in BPF-HiLLIS are realised in the same SDN-like manner as in the implementations of BPF-GLI and BPF-HAIR. These functionalities are implemented by the C++ class *BPF_HiLLIS_Mapping_System*. Each controller component managing a domain creates an object from this class. Thus, this instance represents the mapping system in this domain. Each object representing a node or LAP in the domain keeps a pointer to

the mapping system instance. Moreover, the mapping system instance holds a pointer to each object representing a LAP in the domain. In this way, the mapping system instance has access to the attributes of LAP objects.

A mapping registration packet arriving at a node in a domain is sent to the controller component. After creating a new mapping entry or updating the already existing one, the mapping system instance updates the packet with the corresponding upper level locator of a LAP responsible for the domain. After that, the mapping system instance tells one of the datapaths acting as LAPs in the domain to send the packet via a port interconnected with a node in the domain at the upper level. This node and the mapping system instance of the controller component responsible for the domain at the upper level proceed analogously.

Upon receiving a mapping lookup request packet, a datapath acting as a node sends it to the controller component. The mapping system caches the requested identifier as well as the ID of the sending datapath, the number of its receiving port, and the Ethernet source address of the packet. If a mapping entry is found for the requested identifier, the mapping system instance creates the reply packet and finds the datapath whose ID is cached for the request. After that, the mapping system instance orders the datapath to send the packet via the specified port. Otherwise, the mapping system instance tells one of the datapaths acting as LAPs to send the request packet via one of its ports for the upper level. Because of this SDN-like implementation, the request packet does not have to contain the network address of the requesting endpoint anymore.

If a datapath acting as a node receives a DHCP request packet, it sends the packet to the controller. The object representing this node creates the reply packet and instructs the associated datapath to send the reply packet via the port through which the request packet has arrived. Due to the SDN-like implementation of the mapping functionalities, the reply packet does not have to contain the network address of the mapping system in the domain within which the node is located.

5.5.3 Flow setup

Before sending application data, the source endpoint S sends a network packet to the destination endpoint D. This network packet contains a certain payload type which signals the flow setup to the destination endpoint. By means of this packet, mapping and identifier resolution flows for the destination endpoint are set up at LAPs on the destination side and at the destination edge node.

Upon receiving the flow setup packet, the destination endpoint acknowledges the flow setup by sending a network packet back to the source endpoint. The acknowledgment packet has the same payload type as the flow setup packet. Thus, mapping and identifier resolution flows for the source endpoint are set up at LAPs on the source side and at the source edge node. After receiving the acknowledgment packet, the source endpoint can begin to send application data.

If the source endpoint is finished with its application data, it can send a further network packet to the destination endpoint. The payload type of this packet signals to delete mapping and identifier resolution flows for both endpoints at LAPs on the route as well as at the source and destination edge nodes.

5.6 Testbed

The implementation of BPF-HiLLIS has been deployed in a software testbed realised by the network emulator Mininet [Min16] on a PC with eight Intel Xeon 2.00 GHz CPUs. In this testbed, we run 26 OLV-OpenFlow compatible datapaths which are interconnected according to the testbed topology illustrated in Figure 5.25. Here, each link between two datapaths

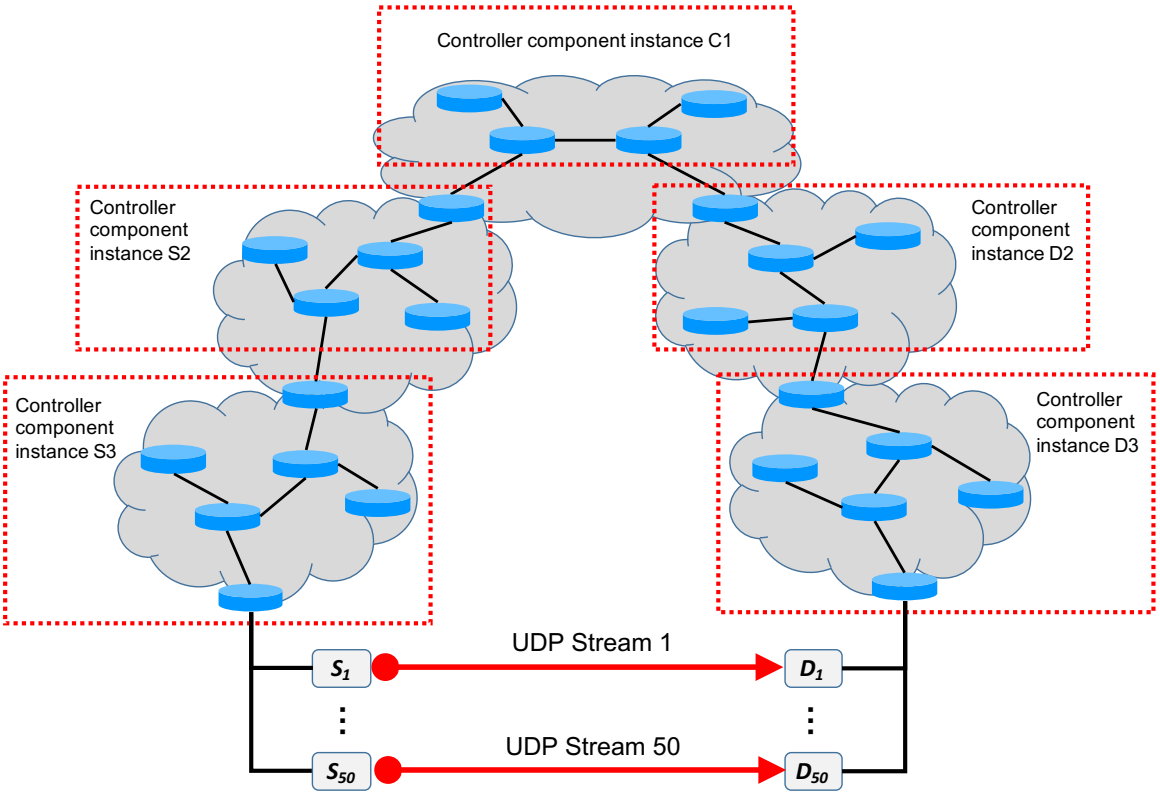


Figure 5.25: Testbed topology for BPF-HiLLIS.

or between a host and a datapath has a bandwidth of 1000 Mbps. Moreover, each network interface of a host and datapath has a MTU of 9000 bytes and a queue length of 1000 frames.

The topology consists of the Core, two domains at level 2, and two domains at level 3. Each of these domains contains four nodes and is interconnected with its immediate parent domain by a single LAP. Thus, each node maintains six unmasked and masked routing table entries. Moreover, each LAP keeps two unmasked and masked routing tables which respectively have six entries. Each domain is managed by its own controller component instance.

In the testbed topology, we have 50 source and destination endpoints which are connected to the edge nodes in the sibling domains at level 3. Thus, a packet from a source endpoint to a destination endpoint, or vice versa, takes 16 hops. Each endpoint is represented by a instance from *BPF-HiLLIS-BNS*. At each source endpoint, a sender runs, while a receiver runs at each destination endpoint. A sender starts an UDP stream to the receiver at a specified destination endpoint. Each stream between 50 source and destination endpoints takes the same route and has a duration of 60 seconds. In an UDP stream, a new UDP datagram is sent every ten milliseconds. Thus, we have a sending rate of 100 Hz which is more than enough for a real-time media communication [ZP13]. Each UDP datagram has a payload size of 8000 bytes carried by an Ethernet jumbo frame.

5.7 Evaluation

In our implementation, a cleartext identifier and locator consists of 16 bytes. Thus, the benchmarks for the cryptographic operations are the same ones as in the implementations of BPF-GLI and BPF-HAIR, which are already presented in Table 3.1. For masked identifier resolution and umasked/masked routing, we have used the same structures as in the implementations of BPF-GLI and BPF-HAIR. The sizes of these structures are already given in Table 4.2. Due to the hierarchical scheme and SDN-like realisation, we have achieved similar routing convergence times as in BPF-HAIR (see Table 4.5).

Table 5.2 shows the sizes of semi- and fully masked structures implemented for BPF-HiLLIS. Unlike in BPF-HAIR, the sizes of endpoint's semi- and fully masked addresses do not depend on the level of the domain in which the endpoint is located. This is because the current level locator is replaced with the next level locator at each domain border.

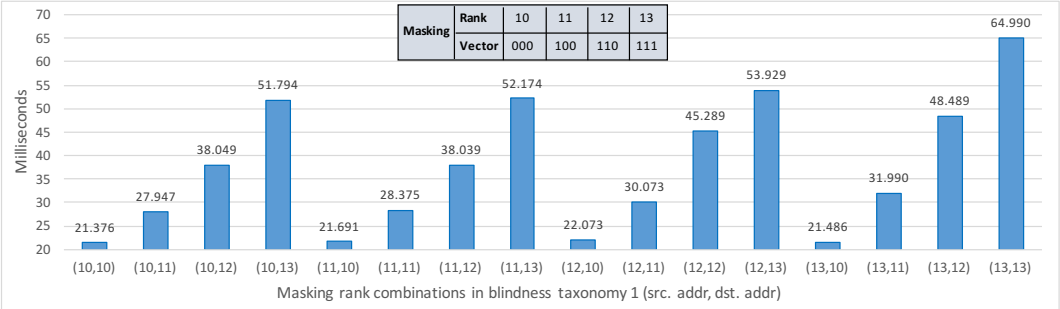
	Size in bytes	
	Semi-masked	***Fully masked***
Address	210	387
Network header	437	791
Mapping registration packet	279	456
Mapping lookup request packet	197	197
Mapping lookup reply packet	283	460
Mapping table entry	288	465
Mapping lookup cache entry	306	483

Table 5.2: Sizes of semi- and fully masked structures.

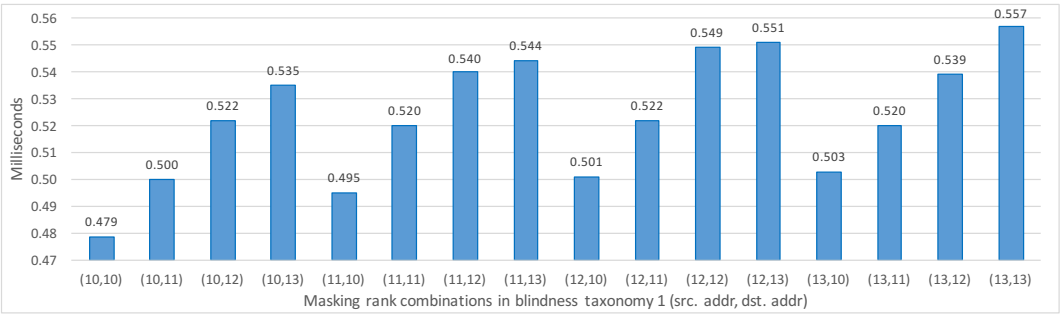
By means of multiple scenarios, the performance of BPF-HiLLIS's implementation has been evaluated in the testbed topology described in Section 5.6. In the first scenario, only the endpoints S_1 and D_1 communicate with each other by means of the masking rank combinations in blindness taxonomies 1 and 2 as well as their cross combinations. For each masking rank combination, S_1 starts a single UDP stream to D_1, and only one UDP stream runs in the network. Each stream has a duration of 60 seconds and a fixed sending rate of 100 Hz. Thus, S_1 sends 6000 packets to D_1 for each masking rank combination. In this regard, an UDP stream is comparable with a directed flooding of the network with UDP datagrams.

Figures 5.26, 5.27, and 5.28 show the flow setup times and average one-way delays (OWDs) in the first scenario. The flow setup times represent the sum of mapping lookup times and OWDs of the flow setup packets for the masking rank combinations. These packets are sent to the controller at the LAPs on the destination side. After mapping lookup and setting up mapping flows for the destination endpoint, semi- or fully masked packet forwarding is performed at the controller on the basis of the masking type for the destination address at the next level. Thus, the masking rank of the destination address is crucial for the flow setup time: the higher the destination masking rank, the more the flow setup time. Moreover, each flow setup packet is also sent to the controller at the destination edge node in order to perform masked identifier resolution and to set up associated flows for the destination endpoint.

After the flow setup, packets carrying user payloads are forwarded on the basis of the constructed flows. In this way, the packets do not have to be sent to the controller anymore. This introduces a significant performance increase in comparison with the implementations of BPF-GLI and BPF-HAIR. Thus, the average OWD for each masking rank combination is not more than 0.6 milliseconds. Here, the masking rank for the destination address of a packet is also crucial for its OWD. This is because more bytes have to be compared and replaced for

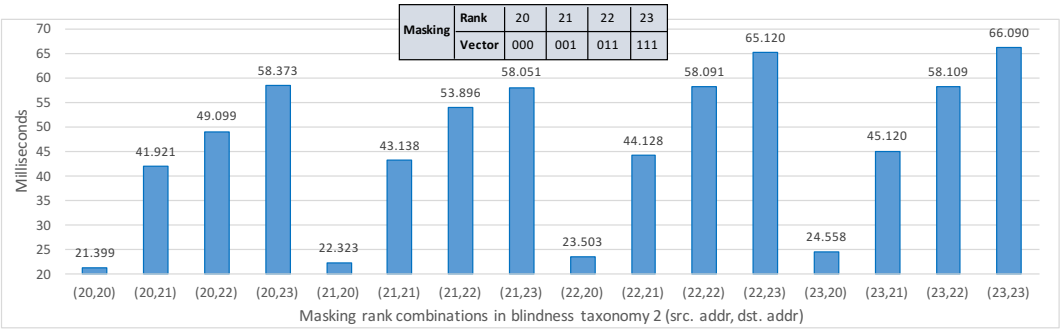


(a) Flow setup times.

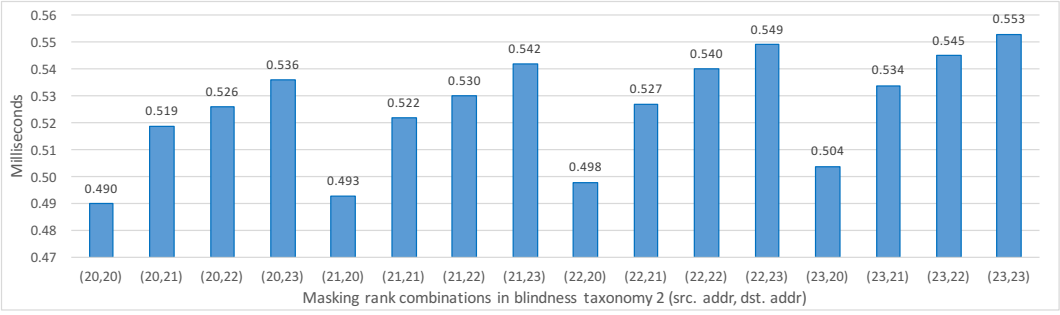


(b) Average one-way delays.

Figure 5.26: Masking rank combinations in blindness taxonomy 1.

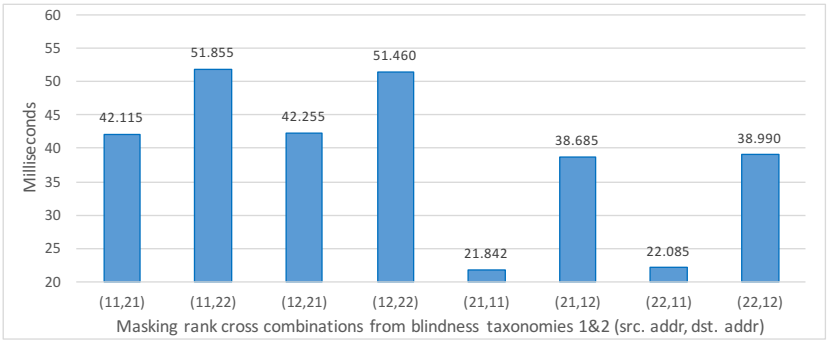


(a) Flow setup times.

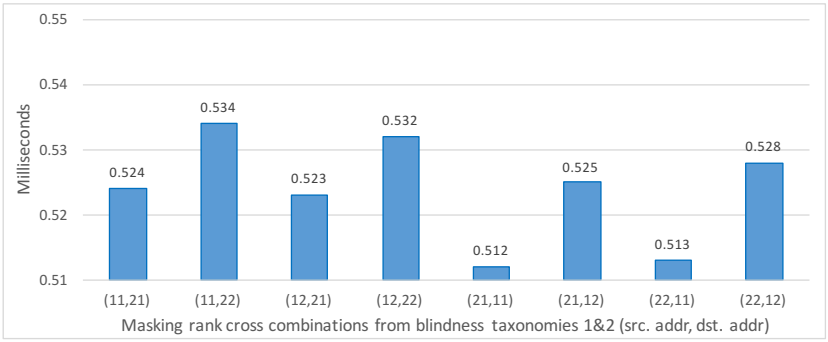


(b) Average one-way delays.

Figure 5.27: Masking rank combinations in blindness taxonomy 2.



(a) Flow setup times.

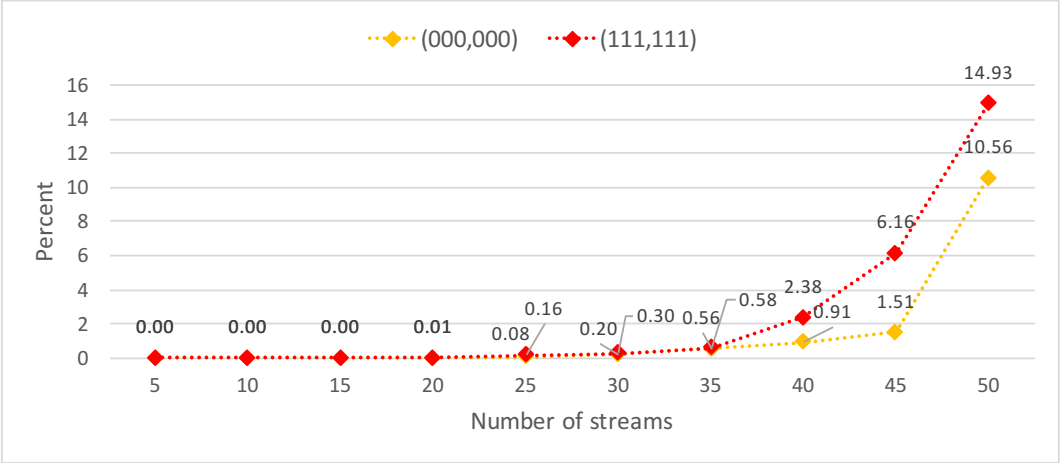


(b) Average one-way delays.

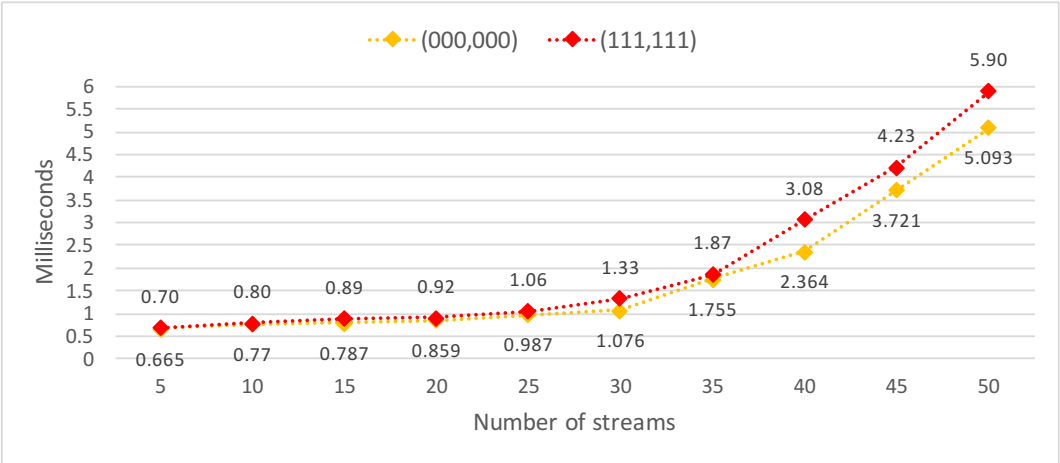
Figure 5.28: Masking rank cross combinations from blindness taxonomies 1 & 2.

a fully masked address than for a semi-masked address. But the differences of the average OWDs for the masking rank combinations vary only within microsecond ranges. Hence, we can state that security levels provided by the masking rank combinations introduce similar overheads regarding the average OWDs. Moreover, the jitter of packet OWDs is also in microsecond ranges. Furthermore, the average throughput for each masking rank combination is 6.4 Mbps.

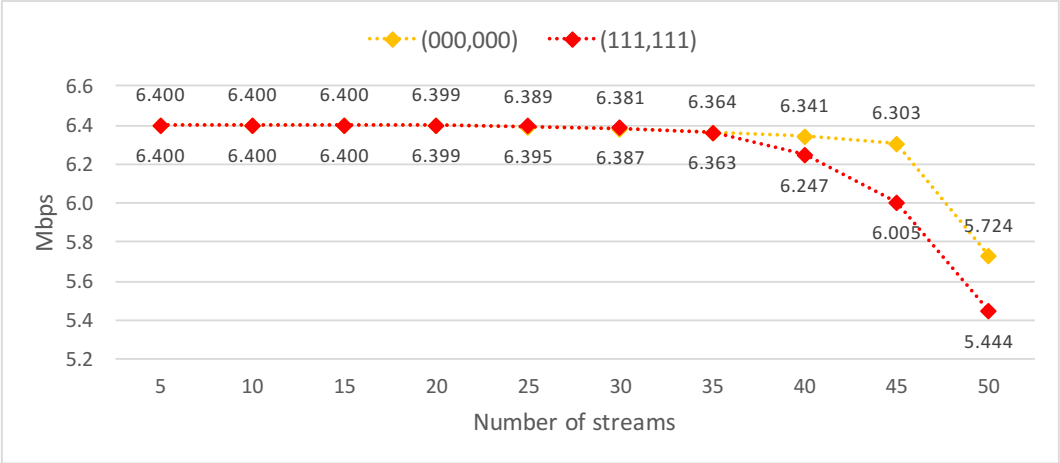
In the second scenario, the network is stressed by up to 50 streams between the source and destination endpoint pairs (S_1, D_1),...,(S_{50}, D_{50}). Here, each stream is masked by means of the weakest and strongest masking vector combinations $(000, 000)$ and $(111, 111)$. In this scenario, we have evaluated the properties of the stream between S_1 and D_1, which are given in Figure 5.29. Packet loss starts from 20 streams and increases exponentially. Packet loss happens, since the queues for the network interfaces of the datapaths begin to be full. For the same reason, the average OWD also increases exponentially. The average throughput stays the same up to 20 streams and then decreases exponentially. For the other masking vector combinations, the stream behaves in the same manner.



(a) Packet losses.



(b) Average one-way delays.



(c) Average throughputs.

Figure 5.29: Stream properties in a network stressed by up to 50 streams.

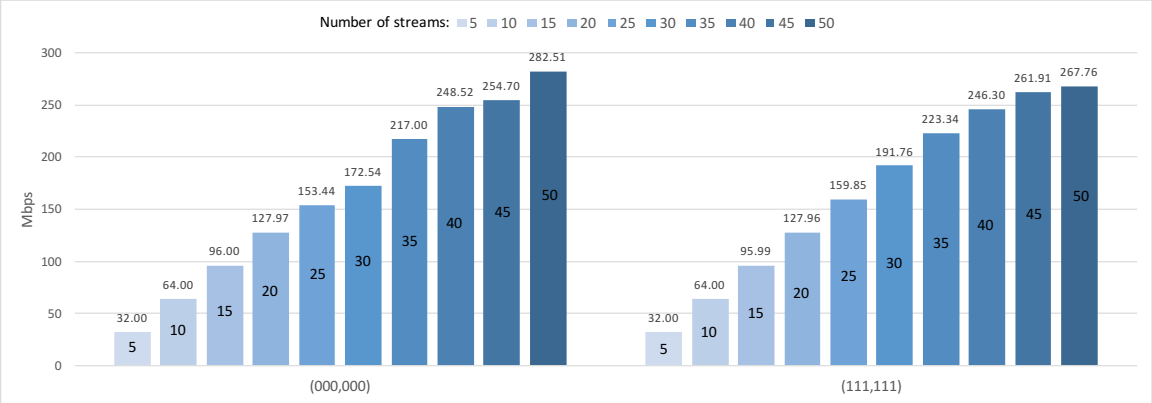


Figure 5.30: Carried load for up to 50 streams.

In the last scenario, we have evaluated the carried load, i.e. the sum of the average throughputs for up to 50 streams between the endpoint pairs (S_1, D_1),...,(S_{50}, D_{50}). As in the previous scenario, each stream here is also masked by means of the weakest and strongest masking vector combinations $(000, 000)$ and $(111, 111)$. As illustrated in Figure 5.30, the carried load firstly increases up to 20 streams linearly but then only logarithmically. This is also because packet loss begins from 20 streams. For the other masking vector combinations, the carried load behaves the same. During the entire evaluation, the system load average[2] was not more than 5.15 corresponding to a CPU utilisation of 64.375% on the testbed PC with eight cores. In summary, we can state that BPF-HiLLIS's implementation achieves a high performance by leveraging flow-based packet forwarding in its entirety. In this way, the implementation of BPF-HiLLIS can even support multiple real-time media communications each with a high sending rate of 100 Hz.

5.8 Conclusion

BPF-HiLLIS combines the beneficial aspects of BPF-GLI and BPF-HAIR. In this way, it fulfills all of the architectural requirements for an adequate BPF design. Moreover, BPF-HiLLIS provides an enhanced masked routing which makes it possible to apply both the semi- and fully blind mode in a domain. Furthermore, this chapter has presented an approach which enables a selective masked routing table entry setup so that the fully blind packet forwarding can be performed on demand.

In BPF-HiLLIS, the masking vectors in a packet determine the blindness modes for the packet

[2]On Unix-like systems, the system load average is a measurement of the computational work performed by the system.

addresses level by level. The blindness level provided by a masking vector is specified by its associated masking rank. Thus, up to 2^x masking ranks can be applied to the endpoint X being located in a domain at level x. We have defined two blindness taxonomies into which certain masking ranks are classified. In blindness taxonomy 1, the higher the masking rank being applied to X's address, the bigger is the radius of domains within which the address is fully masked, beginning with the Core. By applying higher masking ranks in blindness taxonomy 2, the endpoint X can mask its respective location within higher parent domains in direction of the Core. Each of these taxonomies provides $(x + 1)$ different masking ranks for the endpoint X. In each taxonomy, up to $((s + 1) \times (d + 1))$ different masking rank combinations are possible for the source and destination endpoints S and D being located in sibling domains at levels s and d. Moreover, cross combinations of the masking ranks from both blindness taxonomies are also possible. Thus, BPF-HiLLIS supports multiple blindness levels providing different NAC levels which can be flexibly adapted to certain use cases.

In order to leverage the flow-based packet forwarding in its entirety, we have replaced the TLV-based mechanism in OpenFlow with an OLV-based approach, which we have called OLV-OpenFlow. Moreover, we have adapted the implementation of the OpenFlow controller NOX to this OLV-based proceeding. This version of NOX has been called OLV-NOX. For realising the BPF-HiLLIS functionalities on the network side, we have expanded the OLV-OpenFlow controller OLV-NOX by a further component. On the host side, we have extended the framework Blind Network Stack (BNS) to BPF-HiLLIS-BNS. The implementation of BPF-HiLLIS has been deployed in a software testbed realised by the network emulator Mininet.

The performance of BPF-HiLLIS's implementation has been evaluated by means of multiple scenarios. In the first scenario, we have evaluated the flow setup times and average OWDs for masking rank combinations in blindness taxonomies 1 and 2 as well as their cross combinations in an unstressed network. Here, the average OWD for each masking rank combination was not more than 0.6 milliseconds, which represents a significant performance increase in comparison with the implementations of BPF-GLI and BPF-HAIR. Additionally, the differences of the average OWDs for the masking rank combinations have varied only within microsecond ranges. Thus, we can state that security levels provided by the masking rank combinations introduce similar overheads regarding the average OWDs. Moreover, the jitter of packet OWDs was also in microsecond ranges. Furthermore, the average throughput for each masking rank combination was 6.4 Mbps.

In the second scenario, the network has been stressed by up to 50 streams between 50 endpoint pairs. Here, each stream had a fixed sending rate of 100 Hz and has been masked by means of

the weakest and strongest masking vector combinations. In this scenario, we have evaluated the stream properties. Packet loss has started from 20 streams and increased exponentially, since the queues for the network interfaces of the datapaths have begun to be full. For the same reason, the average OWD has also increased exponentially. The average throughput has stayed the same up to 20 streams and then decreased exponentially. In the last scenario, we have evaluated the carried load for up to 50 streams masked by means of the weakest and strongest masking vector combinations. The carried load has firstly increased up to 20 streams linearly but then only logarithmically. This was also because packet loss has begun from 20 streams.

In summary, we can state that BPF-HiLLIS fulfills all of the architectural requirements for an adequate BPF design and provides a fine-grained, flexible and dynamic blindness supporting multiple NAC levels. Moreover, BPF-HiLLIS is a high performance network protocol which can even support multiple real-time media communications each with a high sending rate of 100 Hz.

Chapter 6

Conclusion and outlook

In this thesis, we have presented the Blind Packet Forwarding (BPF) allowing to simultaneously provide network address confidentiality (NAC) and to establish the packet forwarding service. NAC supplies confidentiality for the network addresses of the packets transferred between two communicating endpoints, where only both endpoints are the authorised entities to access the packet addresses in cleartext. Thus, confidentiality for packet addresses is provided during transmission as well as during processing by network nodes.

To demonstrate the general feasibility of our approach, Chapter 3 has presented a basic BPF construction that redesigns the packet forwarding and its associated services to blind ones which can still correctly process masked network addresses being based on a basic structure. In this chapter, we have discussed NAC and its deduced security properties – sender/recipient and relationship unlinkability against different adversary models – local adversary, weak global adversary, and strong global adversary. Moreover, we have declared the effects of applying BPF in the current Internet architecture. This chapter has also discussed the prototype implementation of the basic BPF design. For the implementation on the network side, we have utilised OpenFlow. On the host side, we have created a framework called Blind Network Stack (BNS) in Linux that implements a basic network stack with a socket-like interface in the user space. The prototype implementation of the basic BPF design has been deployed in a real hardware testbed to demonstrate its feasibility for practical deployment capability. Furthermore, this chapter has evaluated the prototype implementation of the basic BPF design.

The basic BPF design and its implementation demonstrate the general feasibility of our approach and its practical deployment capability, while introducing a considerable amount of overhead. Moreover, the basic BPF design is not suitable to be deployed in the current Internet architecture (see Section 3.2.3). Therefore, Chapter 4 has defined the requirements

to be fulfilled by a suitable architecture for BPF. In order to achieve the features for an adequate BPF design, we have used the Loc/ID split principle. Here, we have chosen the approaches GLI-Split and HAIR relying on this principle and extended the basic BPF design to BPF-GLI and BPF-HAIR.

BPF-GLI fulfils the architectural requirements with the exception of a fine-grained hierarchical architecture so that performing super- and subnetting is not always possible in its entirety. This BPF extension supports three modes of blindness providing different NAC levels. While the semi-blindness only provides network identifier confidentiality (NIC), the full blindness provides both NIC and network locator confidentiality (NLC). In case of intra-domain communication, the alternately blind mode only offers NIC as in the semi-blind mode, while NIC and domain-to-domain NLC are supplied in case of inter-domain communication. The sender/recipient and relationship unlinkability properties of these three blindness modes have been discussed against local, weak and strong global adversaries.

BPF-HAIR fulfils all architectural requirements for an adequate BPF design, while the length of a locator sequence for an endpoint is dependent on the level of the domain in which the endpoint is located. Moreover, attaching endpoints to nodes at an arbitrary level is not defined in BPF-HAIR. This BPF extension also defines semi- and full blindness modes which provide the same NAC levels as in BPF-GLI.

Chapter 4 has also presented the prototype implementations of both BPF extensions. Here, we have leveraged the SDN-like realisation and flow-based forwarding for the implementations of both BPF extensions. On the host side, we have extended the framework BNS to BPF-GLI-BNS and BPF-HAIR-BNS. The prototype implementations of both BPF extensions have been deployed in a real hardware testbed to demonstrate their feasibility for practical deployment. Furthermore, this chapter has evaluated the implementations of both BPF extensions.

While both BPF extensions demonstrate that BPF can in principle be realised by means of the existing FNA approaches, neither of the BPF extensions can achieve all of the properties required for an adequate BPF design. Therefore, Chapter 5 has combined the beneficial aspects of both BPF extensions into a new design called Blind Packet Forwarding in Hierarchical Level-based Locator/Identifier Split (BPF-HiLLIS). BPF-HiLLIS provides an enhanced masked routing which makes it possible to apply both the semi- and fully blind mode in a domain. Furthermore, this chapter has presented an approach which enables a selective masked routing table entry setup so that the fully blind packet forwarding can be performed on demand.

BPF-HiLLIS provides a fine-grained, flexible and dynamic blindness providing up to 2^x masking ranks for the endpoint X being located in a domain at level x. We have defined two blindness taxonomies into which certain masking ranks are classified. In blindness taxonomy 1, the higher the masking rank being applied to X's address, the bigger is the radius of domains within which the address is fully masked, beginning with the Core. By applying higher masking ranks in blindness taxonomy 2, the endpoint X can mask its respective location within higher parent domains in direction of the Core. Each of these taxonomies provides $(x + 1)$ different masking ranks for the endpoint X. Thus, BPF-HiLLIS supports multiple blindness levels providing different NAC levels which can be flexibly adapted to certain use cases.

In order to leverage the flow-based packet forwarding in its entirety, we have replaced the TLV-based mechanism in OpenFlow with an OLV-based approach, which we have called OLV-OpenFlow. Moreover, we have adapted the implementation of the OpenFlow controller NOX to this OLV-based proceeding. This version of NOX has been called OLV-NOX. For realising the BPF-HiLLIS functionalities on the network side, we have expanded the OLV-OpenFlow controller OLV-NOX by a further component. On the host side, we have extended the framework BNS to BPF-HiLLIS-BNS. The implementation of BPF-HiLLIS has been deployed in a software testbed realised by the network emulator Mininet. The performance of BPF-HiLLIS's implementation has been evaluated by means of multiple scenarios. On the basis of the evaluations, we have stated that BPF-HiLLIS is a high performance network protocol which can even support multiple real-time media communications each with a high sending rate.

6.1 Outlook

PEKS gives the ability to blindly but still correctly compare encrypted data with each other without revealing their cleartext contents. In this thesis, we have discussed that PEKS can be used to realise a new kind of end-to-end confidentiality for data processed by third parties, where only the data owner is classified as authorised entity having access to the data in cleartext. As discussed in Section 2.1, PEKS has been already applied in multiple approaches. In contrast to these approaches focusing on high-level services, BPF has tackled the packet forwarding service in order to provide NAC. Network packet addresses are the basic information transferred between two communicating endpoints, and the packet forwarding service is the fundamental in-network service needed by the endpoints.

NAC has not been tackled only by BPF. IP Encapsulating Security Payload (ESP) protocol

in tunnel mode also focuses on the confidentiality for the ultimate source and destination addresses of a packet within the public network (see Section 3.2.1.1). But NAC is not provided within private networks, and security gateways still have access to the addresses in cleartext. Moreover, the addresses of the security gateways in the outer header are transferred in cleartext so that NAC is not supplied for these addresses. In this regard, BPF is the first approach providing NAC in its entirety. Additionally, BPF, due to its traffic flow confidentiality, hides potential targets against active attacks such as address spoofing, man-in-the-middle and repetition attack. However, BPF defines a new network architecture and addressing structure. Therefore, deploying BPF in the Internet needs the support of network providers and can thus take time as well as effort.

As already discussed in Section 3.2, anonymity of a subject and confidentiality of information, which can be used to identify the subject, are related to each other. In this regard, we have stated that NAC deduces the anonymity properties – sender/recipient and relationship unlinkability introduced in Section 2.4.1. In Section 3.2.1, we have discussed multiple prominent low-latency anonymity systems aiming to provide these unlinkability properties as well as their vulnerabilities. Section 3.2.2 has stated that BPF with regard to sender/recipient and relationship unlinkability is resistant to a strong adversary model. Thus, BPF can be also regarded as a low-latency anonymity system supplying these crucial security properties. In contrast to other low-latency anonymity systems to be deployed in the current Internet architecture, BPF, however, requires a new network architecture.

Beside the security properties above, information integrity is also crucial. However, BPF does not aim to provide integrity of network packet addresses, routing update messages, and DHCP reply messages. In Sections 3.2.2 and 4.1.4.5, we have discussed how to extend BPF in order to also provide integrity for packet addresses as well as routing update and DHCP reply messages.

For the implementation of BPF functionalities on the network side, we have leveraged the SDN-like realisation by using OpenFlow (see Section 4.1.5.1). In this way, we have demonstrated that the networks and functionalities can be bundled together, which improves the performance and makes the network configuration more flexible and dynamic. Here, the routing functionalities are crucial. Although we have just implemented a decentralised routing algorithm in central manner, we have achieved a significant performance increase for the routing table setup (see Tables 4.3 and 4.5). In this regard, the existing routing algorithms can be adapted to the SDN-like realisation, but also new routing algorithms acting in central manner can be developed.

For realising the BPF functionalities on the host side, we have implemented the framework BNS in the user space (see Sections 3.3.2 and 4.1.5.2). In order to boost BNS's performance, it can be integrated into the Linux kernel. BNS implements only UDP functionalities. Here, BNS can be expanded by the functionalities of other transport protocols such as TCP.

In order to achieve a BPF implementation which provides high performance, we have extended OpenFlow to OLV-OpenFlow (see Section 5.4). This OpenFlow extension makes it possible that clean-slate designs, which define another packet structure than the IP packet structure, can leverage the flow-based packet forwarding in its entirety. In order to demonstrate the feasibility of this approach, we have implemented the OLV-OpenFlow functionalities in a user-space software switch. Here, OLV-OpenFlow can be realised in a hardware switch and evaluated with respect to its performance.

On the basis of multiple scenarios, BPF's implementations have basically been evaluated in software as well as hardware testbeds set up in a laboratory environment. Before deploying BPF in a customer environment like the Internet, it is highly recommended to widely evaluate BPF in extensive research environments such as German-Lab [SRZ+14]. Here, ToMaTo [MSC14] can be used to manage the experiment components with regard to high realism.

As stated above, BPF is just the beginning and provides a solid base for future research towards achieving a new state "*establishing smart in-network services* ***as well as*** *providing information confidentiality*". On the basis of BPF, further in-network services requiring access to control data at different layers can be redesigned to blind ones which can still correctly process control data in encrypted form. Since BPF handles packet addresses in encrypted form and packet addresses act as the basis for operations performed by further in-network services, a redesign of the services is highly recommended. Here, Deep Packet Inspection systems give prominence to the importance of handling control data in encrypted form, since information needed by these systems are more user-specific. Moreover, BPF is a clean-slate approach supplying significant anonymity properties against a strong adversary model. There arises the question whether, if even possible, how to achieve these properties or a part of them without a clean-slate proceeding.

List of Figures

List of Tables

Bibliography

[AB12] RJ Atkinson and SN Bhatti. Identifier-Locator Network Protocol (ILNP) Architectural Description. Standards Track RFC 6740, IETF, November 2012. 2.2

[ALPK07] Adam J Aviv, Michael E Locasto, Shaya Potter, and Angelos D Keromytis. Ssares: Secure searchable automated remote email storage. In *Computer Security Applications Conference, 2007. ACSAC 2007. Twenty-Third Annual*, pages 129–139. IEEE, 2007. 2.1, 3.3, 4.1.5, 4.2.4, 5.5

[BDCOP04] Dan Boneh, Giovanni Di Crescenzo, Rafail Ostrovsky, and Giuseppe Persiano. Public key Encryption with Keyword Search (PEKS). In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 506–522. Springer, 2004. 2.1

[BF01] Dan Boneh and Matt Franklin. Identity-based encryption from the Weil pairing. In *Annual International Cryptology Conference*, pages 213–229. Springer, 2001. 2.1

[BG87] Dimitri P Bertsekas and Robert G Gallager. Distributed asynchronous bellman-ford algorithm. *Data networks*, page 4, 1987. 3.1.3

[BSNS08] Joonsang Baek, Reihaneh Safavi-Naini, and Willy Susilo. Public key encryption with keyword search revisited. In *International conference on Computational Science and Its Applications*, pages 1249–1259. Springer, 2008. 2.1

[Cha81] David L Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 24(2):84–90, 1981. 2.4, 3.2.1

[CPq16a] CPqD. nox13oflib, 2016. URL: https://github.com/CPqD/nox13oflib. 5.4.2

[CPq16b] CPqD. ofsoftswitch13, 2016. URL: https://github.com/CPqD/ofsoftswitch13. 5.4.2

[Dan03] George Danezis. Mix-networks with restricted routes. In *International Workshop on Privacy Enhancing Technologies*, pages 1–17. Springer, 2003. 2.4, 3.2.1

[DC07] George Danezis and Richard Clayton. *Introducing traffic analysis.* Auerbach Publications, Boca Raton, FL, 2007. 2.4.6, 3.2, 3.2.1.2

[DH98] Stephen E. Deering and Robert M. Hinden. Internet Protocol, Version 6 (IPv6). Standards Track RFC 2460, IETF, December 1998. 1

[Die08] Tim Dierks. The transport layer security (TLS) protocol version 1.2. Standards Track RFC 5246, IETF, August 2008. 1

[DMS04] Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: The second-generation onion router. Technical report, DTIC Document, 2004. 2.4.3, 2.10, 2.11, 3.2.1.2, 6.1

[DR13] Joan Daemen and Vincent Rijmen. *The design of Rijndael: AES-the advanced encryption standard.* Springer Science & Business Media, 2013. 1

[EY09] Matthew Edman and Bülent Yener. On anonymity in an electronic society: A survey of anonymous communication systems. *ACM Computing Surveys (CSUR)*, 42(1):5, 2009. 2.4, 3.2.1

[FCM+09] Anja Feldmann, Luca Cittadini, Wolfgang Mühlbauer, Randy Bush, and Olaf Maennel. HAIR: Hierarchical architecture for Internet routing. In *Proceedings of the 2009 workshop on Re-architecting the internet*, pages 43–48. ACM, 2009. 2.2, 2.2.2, 2.4, 2.5, 4.2, 6.1

[FFML13] Dino Farinacci, Vince Fuller, Dave Meyer, and Darrel Lewis. The Locator/ID Separation Protocol (LISP). Standards Track RFC 6830, IETF, January 2013. 2.2

[FM02] Michael J Freedman and Robert Morris. Tarzan: A peer-to-peer anonymizing network layer. In *Proceedings of the 9th ACM conference on Computer and communications security*, pages 193–206. ACM, 2002. 2.4.5, 2.13, 3.2.1.4, 6.1

[Fun12] Open Networking Fundation. Software-defined networking: The new norm for networks. *ONF White Paper*, 2012. 2.3, 2.6, 6.1

[GKP+08] Natasha Gude, Teemu Koponen, Justin Pettit, Ben Pfaff, Martín Casado, Nick McKeown, and Scott Shenker. NOX: towards an operating system for networks. *ACM SIGCOMM Computer Communication Review*, 38(3):105–110, 2008. 2.3.3

[HBLL99] Mor Harchol-Balter, Tom Leighton, and Daniel Lewin. Resource discovery in distributed networks. In *Proceedings of the eighteenth annual ACM symposium on Principles of distributed computing*, pages 229–237. ACM, 1999. 2.4.5.1

[Hed88] Charles L Hedrick. Routing Information Protocol. Standards Track RFC 1058, IETF, June 1988. 3.1.3

[Hew16] Hewlett Packard (HP). HP 3800-24G-2SFP+ switch, 2016. URL: http://www8.hp.com/de/de/products/networking-switches/product-detail.html?oid=5171789. 3.4

[HMH13] Michael Hoefling, Michael Menth, and Matthias Hartmann. A survey of mapping systems for locator/identifier split Internet routing. *IEEE Communications Surveys & Tutorials*, 15(4):1842–1858, 2013. 2.2

[HSK10] Christian Henke, Abbas Siddiqui, and Rahamatullah Khondoker. Network functional composition: State of the art. In *Telecommunication Networks and Applications Conference (ATNAC), 2010 Australasian*, pages 43–48. IEEE, 2010. 1

[HSKE09] Oliver Hanka, Christoph Spleiß, Gerald Kunzmann, and Jörg Eberspächer. A novel DHT-Based network architecture for the Next Generation Internet. In *Networks, 2009. ICN'09. Eighth International Conference on*, pages 332–341. IEEE, 2009. 2.2

[HYH13] Shih-Ting Hsu, Chou Chen Yang, and Min-Shiang Hwang. A Study of Public Key Encryption with Keyword Search. *IJ Network Security*, 15(2):71–79, 2013. 2.1

[Ken05] Stephen Kent. IP Encapsulating Security Payload (ESP). Standards Track RFC 4303, IETF, December 2005. 1, 2.4.2, 3.2.1, 3.2.1.1

[Kha06] Dalia Khader. Public key encryption with keyword search based on K-resilient IBE. In *International Conference on Computational Science and Its Applications*, pages 298–308. Springer, 2006. 2.1

[KS05] Stephen Kent and Karen Seo. Security Architecture for the Internet Protocol. Standards Track RFC 4301, IETF, December 2005. 1

[KTB+07] Csaba Kiraly, Simone Teofili, Giuseppe Bianchi, Renato Lo Cigno, Matteo Nardelli, and Emanuele Delzeri. Traffic flow confidentiality in IPsec: Protocol and implementation. In *IFIP International Summer School on the Future of Identity in the Information Society*, pages 311–324. Springer, 2007. 3.2.1.1

[KXNP08] E-yong Kim, Li Xiao, Klara Nahrstedt, and Kunsoo Park. Secure Interdomain Routing Registry. *IEEE Transactions on Information Forensics and Security*, 3(2):304–316, 2008. 2.1

[Li11] Tony Li. Recommendation for a Routing Architecture. Standards Track RFC 6115, IETF, February 2011. 2.2, 4

[LLD09] IEEE Standard for Local and Metropolitan Area Networks– Station and Media Access Control Connectivity Discovery. *IEEE Std 802.1AB-2009 (Revision of IEEE Std 802.1AB-2005)*, pages 1–204, Sept 2009. 3.1.2

[LRWW04] Brian N Levine, Michael K Reiter, Chenxi Wang, and Matthew Wright. Timing attacks in low-latency mix systems. In *International Conference on Financial Cryptography*, pages 251–265. Springer, 2004. 2.4.6, 3.2.1.2

[LWW09] Qin Liu, Guojun Wang, and Jie Wu. An efficient privacy preserving keyword search scheme in cloud computing. In *Computational Science and Engineering, 2009. CSE'09. International Conference on*, volume 2, pages 715–720. IEEE, 2009. 2.1

[MHH10] Michael Menth, Matthias Hartmann, and Michael Hofling. Firms: A mapping system for future internet routing. *IEEE Journal on Selected Areas in Communications*, 28(8):1326–1331, 2010. 2.2.1.2

[MHK13] Michael Menth, Matthias Hartmann, and Dominik Klein. Global locator, local locator, and identifier split (GLI-Split). *Future Internet*, 5(1):67–94, 2013. 2.2, 2.2.1, 2.1, 2.2, 2.3, 4.1, 6.1

[Min16] Mininet. A network emulator, 2016. URL: http://mininet.org/. 5.6

[MN06] Robert Moskowitz and Pekka Nikander. Host Identity Protocol (HIP) Architecture. Standards Track RFC 4423, IETF, May 2006. 2.2

[MSC14] Paul Müller, Dennis Schwerdel, and Justin Cappos. Tomato a virtual research environment for large scale distributed systems research. *PIK-Praxis der Informationsverarbeitung und Kommunikation*, 37(1):23–32, 2014. 2.3, 6.1

[MZF07] David Meyer, Lixia Zhang, and Kevin Fall. Report from the IAB Workshop on Routing and Addressing. Standards Track RFC 4984, IETF, September 2007. 2.2, 4

[nox16] noxrepo. NOX: A Network Operating System for OpenFlow, 2016. URL: http://archive.openflow.org/downloads/Workshop2009/OpenFlowWorkshop-MartinCasado.pdf. 2.8, 6.1

[Opea] OpenFlow Switch Specification. Version 1.3.0 (Protocol version 0x04), June, 2012. URL: https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.3.0.pdf. 2.3, 2.7, 2.3.1, 6.1

[Opeb] OpenFlow Switch Specification. Version 1.5.1 (Protocol version 0x06), December, 2014. URL: https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-switch-v1.5.1.pdf. 2.3

[Ope16a] Open Networking Foundation (ONF). OpenFlow, 2016. URL: https://www.opennetworking.org/sdn-resources/openflow. 1.2, 2.3

[Ope16b] Open Networking Foundation (ONF). OpenFlow Technical Library, 2016. URL: https://www.opennetworking.org/sdn-resources/technical-library. 2.3

[Ope16c] Open Networking Foundation (ONF). SDN Product Directory, 2016. URL: https://www.opennetworking.org/products-listing. 2.3

[OS06] Lasse Overlier and Paul Syverson. Locating hidden servers. In *2006 IEEE Symposium on Security and Privacy (S&P'06)*, pages 15–pp. IEEE, 2006. 3.2.1.2

[PH05] Andreas Pfitzmann and Marit Hansen. Anonymity, unlinkability, unobservability, pseudonymity, and identity management-a consolidated proposal for terminology. 2005. 2.4, 2.9, 2.4.1, 3.2, 6.1

[Pos80] Jonathan Bruce Postel. User Datagram Protocol. Standards Track RFC 768, IETF, August 1980. 2.4.5.3

[Pos81a] Jonathan Bruce Postel. Internet Protocol. Standards Track RFC 791, IETF, September 1981. 1

[Pos81b] Jonathan Bruce Postel. Transmission Control Protocol. Standards Track RFC 793, IETF, September 1981. 2.3.2

[PPJB08] Jianli Pan, Subharthi Paul, Raj Jain, and Mic Bowman. MILSA: A Mobility and Multihoming Supporting Identifier Locator Split Architecture for Naming in the Next Generation Internet. In *IEEE GLOBECOM 2008-2008 IEEE Global Telecommunications Conference*, pages 1–6. IEEE, 2008. 2.2

[RLH06] Yakov Rekhter, Tony Li, and Susan Hares. A Border Gateway Protocol 4 (BGP-4). Standards Track RFC 4271, IETF, January 2006. 3.1.3

[RPSL10] Hyun Sook Rhee, Jong Hwan Park, Willy Susilo, and Dong Hoon Lee. Trapdoor security in a searchable public-key encryption scheme with a designated tester. *Journal of Systems and Software*, 83(5):763–771, 2010. 2.1

[RR98] Michael K Reiter and Aviel D Rubin. Crowds: Anonymity for web transactions. *ACM Transactions on Information and System Security (TISSEC)*, 1(1):66–92, 1998. 2.4.4, 2.12, 3.2.1.3, 6.1

[RSA78] Ronald L Rivest, Adi Shamir, and Leonard Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978. 1

[RW10] Jian Ren and Jie Wu. Survey on anonymous communications in computer networks. *Computer Communications*, 33(4):420–431, 2010. 2.4, 3.2.1

[SBJR13] Irfan Simsek, Martin Becke, Yves Igor Jerschow, and Erwin P Rathgeb. A clean-slate security vision for future networks: Simultaneously ensuring information security and establishing smart in-network services using the example of blind packet forwarding. In *Fourth International Conference on the Network of the Future (NOF) 2013*, pages 1–5. IEEE, October 2013. 1.2

[SDS02] Andrei Serjantov, Roger Dingledine, and Paul Syverson. From a trickle to a flood: Active attacks on several mix types. In *International Workshop on Information Hiding*, pages 36–52. Springer, 2002. 2.4, 3.2.1

[SG 12] ITU-T SG 13. Identification Framework in Future Networks. Recommendation Y.3031, ITU-T, May 2012. 2.2, 4

[SJBR14] Irfan Simsek, Yves Igor Jerschow, Martin Becke, and Erwin P Rathgeb. Blind Packet Forwarding in a Hierarchical Architecture with Locator/Identifier Split. In *Fifth International Conference and Workshop on the Network of the Future (NOF) 2014*, pages 1–5. IEEE, December 2014. 1.2

[SJR15] Irfan Simsek, Yves Igor Jerschow, and Erwin P Rathgeb. Blind Packet Forwarding Using a Locator/Identifier Split Approach. In *ISCC 2015 - The Twentieth IEEE Symposium on Computers and Communications*, pages 401–408. IEEE, July 2015. 1.2

[SÖM10] Abdullatif Shikfa, Melek Önen, and Refik Molva. Privacy and confidentiality in context-based and epidemic forwarding. *Computer Communications*, 33(13):1493–1504, 2010. 2.1

[SRZ+14] Dennis Schwerdel, Bernd Reuther, Thomas Zinner, Paul Müller, and Phouc Tran-Gia. Future Internet research and experimentation: The G-Lab approach. *Computer Networks*, 61:102–117, 2014. 2.3, 6.1

[Tim96] Brenda Timmerman. Adaptable traffic masking techniques for traffic flow confidentiality on internetworks. *University of Southern California, Technical Report*, 641, 1996. 3.2.2

[VLBA12] Thomas Volkert, Florian Liers, Martin Becke, and Hakim Adhari. Requirements-oriented path selection for multipath transmission. In *Proceedings of the Joint ITG and Euro-NF Workshop on Visions of Future Generation Networks (EuroView), Würzburg, Bayern/Germany*, 2012. 1

[WALS02] Matthew Wright, Micah Adler, Brian Neil Levine, and Clay Shields. An Analysis of the Degradation of Anonymous Protocols. In *NDSS*, volume 2, pages 39–50, 2002. 2.4.6, 3.2.1.4, 3.2.2

[WALS04] Matthew K Wright, Micah Adler, Brian Neil Levine, and Clay Shields. The predecessor attack: An analysis of a threat to anonymous communications systems. *ACM Transactions on Information and System Security (TISSEC)*, 7(4):489–522, 2004. 2.4.6, 3.2.1.3

[Wol10] Tilman Wolf. In-network services for customization in next-generation networks. *IEEE Network*, 24(4):6–12, 2010. 1

[XJ09] Xiaohu Xu and R Jain. Routing Architecture for the Next-Generation Internet (RANGI), 2009. 2.2

[YL06] Tatu Ylonen and Chris Lonvick. The secure shell (SSH) protocol architecture. Standards Track RFC 4251, IETF, January 2006. 1

[YS01] Ruixi Yuan and W Timothy Strayer. *Virtual private networks: technologies and solutions*. Addison-Wesley Longman Publishing Co., Inc., 2001. 5.1, 5.3.3.1

[YXZ11] Hao-Miao Yang, Chun-Xiang Xu, and Hong-Tian Zhao. An efficient public key encryption with keyword scheme not using pairing. In *Instrumentation, Measurement, Computer, Communication and Control, 2011 First International Conference on*, pages 900–904. IEEE, 2011. 2.1

[ZCM+12] Yuanjie Zhao, Xiaofeng Chen, Hua Ma, Qiang Tang, and Hui Zhu. A New Trapdoor-indistinguishable Public Key Encryption with Keyword Search. *JoWUA*, 3(1/2):72–81, 2012. 2.1

[ZP13] Xiaoqing Zhu and Rong Pan. Nada: A unified congestion control scheme for low-latency interactive video. In *2013 20th International Packet Video Workshop*, pages 1–8. IEEE, 2013. 5.6